Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Jiří Královec

RoboRally

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.

Studijní program: Informatika

Studijní obor: Programování

Praha 2012

Rád bych touto cestou poděkoval všem mým kolegům, kteří mi radili a konzultovali se mnou technologie a vhodné postupy. Dále děkuji RNDr. Tomáši Holanovi, Ph.D., že se mnou podrobně konzultoval náplň práce, zvolené technologie a poskytl mi cenné rady. Nakonec bych také rád poděkoval mé rodině za jejich podporu.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 1. 8. 2012 podpis

Název práce: RoboRally

Autor: Jiří Královec

Katedra / Ústav: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D., Kabinet software a výuky informatiky

Abstrakt: Práce se zabývá realizací počítačové verze deskové hry RoboRally. Vytvořený program umožňuje hru jednoho nebo více hráčů na jednom počítači. Do hry je možné přidat i počítačem řízené hráče. Program umožňuje sestavení hracího plánu z předem připravených desek, je možné přidávat i nové vlastní desky. Pro realizaci umělé inteligence řídící počítačové hráče byly vytvořeny dva enginy (počítačoví hráči) s různými způsoby rozhodování. Dále byl vytvořen program hledající vhodné nastavení enginů pomocí techniky genetických algoritmů a program umožňující provádět souboje mezi různě nastavenými enginy a tím je porovnávat. Pomocí implementovaných prostředků byla nalezena taková nastavení jednotlivých enginů, která jsou člověku rovnocennými protihráči.

Klíčová slova: RoboRally, desková hra, umělá inteligence, genetické algoritmy

Title: RoboRally

Author: Jiří Královec

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D., Department of Software and Computer Science Education

Abstract: This work deals with the implementation of computer version of the board game RoboRally. The computer program which was created enables game for one or more players on one computer. Computer driven players can also be added into a game. Because of creating the artificial intelligence for driving computer players, two different engines (computer players) were created. In addition another program was created for searching adequate setting of the parameters of the engines. The program conducts the search using the technique called genetic algorithms. Using created tools, good enough settings were found to make created engines equivalent opponents for human players.

Keywords: RoboRally, board game, artificial intelligence, genetic algorithms

Obsah

1 Úvod

Tato práce se zabývá realizací počítačové verze deskové hry RoboRally. Hra RoboRally, vymyšlená již roku 1985, byla poprvé vydána ve stolní verzi v roce 1994. Od té doby vyšla ještě několikrát a její jednotlivá vydání byla postupně doplňována o různá rozšíření. Tato práce byla vytvořena podle vydání z roku 2005. Kromě stolních verzí hry, které jsou ke koupi v obchodní síti, jsou na Internetu dostupné i verze počítačové, které více či méně zachovávají podobu stolní verze co do pravidel i vizuální podoby hry.

Základním cílem práce je vytvoření počítačového programu, který bude umožňovat hraní deskové hry RoboRally na PC (dále většinou jen program). Program bude až na výjimky přesně implementovat pravidla RoboRally. Hrát se bude na jednom počítači, v jednom nebo více hráčích, tedy se předpokládá, že se jednotliví hráči budou u herního počítače střídat. Do hry bude možné přidat i počítačem řízené hráče, s tím souvisí požadavek vytvoření umělé inteligence, která bude srovnatelná s průměrnými lidskými hráči.

Vlastní implementace je realizována se záměrem nejenom vytvořit program umožňující hru jako takovou, ale i otestovat různé prvky umělé inteligence a následně vytvořit nadstavbu umožňující vyhledání optimálního respektive suboptimálního nastavení parametrů algoritmů zajišťujících tuto umělou inteligenci

Text práce je kromě úvodu a závěru členěn do čtyř kapitol. Kapitola 2 Pravidla hry popisuje hru RoboRally a vysvětluje, co a proč bylo v implementaci změněno oproti jejím pravidlům.

Kapitola 3 Realizace systému stručně popisuje, jaké programy byly vytvořeny a jaké nástroje byly k jejich vytvoření použity. Podrobnější popis programů obsahuje příloha B Programátorská dokumentace.

Kapitola 4 Vytvoření počítačového hráče se zabývá vytvořením umělé inteligence schopné rovnocenně hrát s lidskými hráči. Rozebírá možné postupy a ukazuje, které byly nakonec použity.

Kapitola 5 Nastavení parametrů pomocí genetických algoritmů se zabývá nastavením různých parametrů, které ovlivňují chování vytvořené umělá inteligence počítačového hráče. Ukazuje, jak byly vhodné parametry nalezeny pomocí genetických algoritmů.

Kapitola 6 Srovnání s existujícími implementacemi ukazuje jiné implementace hry RoboRally a srovnává je programem vytvořeným v rámci této bakalářské práce.

Nedílnou součástí předloženého textu jsou přílohy, které obsahují Uživatelskou dokumentaci (příloha A), Programátorskou dokumentaci (příloha B) a informace o obsahu přiloženého CD (příloha C).

2 Pravidla hry

RoboRally je desková hra, kterou mohou hrát minimálně dva a maximálně osm hráčů. Před zahájením hry se nejdříve sestaví herní plán. Každý hráč ovládá jednoho robota. Hra probíhá po kolech, každé kolo je rozděleno na dvě části – programovací a prováděcí. V programovací části všichni hráči najednou programují své roboty, v prováděcí části pak všichni roboti najednou vykonávají, co mají naprogramováno. Pohyb každého robota po herním plánu je určován předně posloupností kroků (programem) stanovených příslušným hráčem a následně ovlivněn ovládacími prvky (políčky) herního plánu, na kterých se robot nachází. Cílem každého hráče je jako první provést svého robota skrze políčka, na kterých jsou umístěné vlajky a to ve správném pořadí. Pravidla hry jsou převzata z RoboRally rulebook [1], návodu ke hře, který je přiložen v každém balení hry vydané v roce 2005.

2.1 Hrací plán

Hrací plán se skládá z hracích desek. V balení hry zakoupené v obchodě je k dispozici celkem 8 různých hracích desek, všechny tyto desky jsou čtvercové o velikosti dvanáct krát dvanáct políček. Plán se vytvoří pomocí alespoň jedné takové desky. Pokud je desek ve hře více, skládají se vedle sebe tak, že tvoří souvislé hrací pole a sousedící desky se dotýkají vždy celou stranou. K takto vytvořenému hracímu plánu je vždy třeba přiložit speciální desku, obsahující startovní pozice pro roboty. Ta má jednu stranu dlouhou dvanáct políček, která obsahuje startovní pozice pro jednotlivé roboty. Hrací plán se sestavuje před zahájením hry a může být pro každou hru jiný. Pak zůstává po celou dobu hry stejný a vidí jej celý všichni hráči.

Jedním ze základních požadavků na program realizující uvedenou hru je umožnění vytvářet jednotlivé hrací desky a přidávat je do programu. Z nich musí jít před hrou vytvářet hrací plány. Bylo by také vhodné, aby šlo již vytvořené hrací plány uložit a později je použít v další hře, případně mít již po nainstalování programu možnost použít předpřipravené celé hrací plány.

2.2 Políčka

Každá hrací deska se skládá z jednotlivých stejně velkých čtvercových políček. Ta jsou vedle sebe poskládána stejně jako na šachovnici, s tím rozdílem, že strana každé desky má dvanáct políček (kromě startovní desky). Každé z těchto políček má

na roboty ve hře jiný vliv, který se navíc uplatní jen v určitý čas. Druhy políček jsou následující:

1) Obyčejné políčko – neovlivňuje na něm stojícího robota.
2) Dopravník – posune robota o jedno políčko ve směru svých šipek.
3) Expresní dopravník – jako dopravník, ale je aktivní dvakrát během každého registru.
4) Převodovka – otočí robota, který na ní stojí o 90° ve směru svých šipek.
5) Bezedná jáma – robot, který se na ni dostane, zemře.
6) Opravna – robotu stojícímu na opravně na konci kola, bude opraveno jedno poškození.
7) Vylepšující opravna – jako opravna, robot navíc dostane jednu kartu s vylepšením.

Políčko může navíc obsahovat další prvky. Na každé straně políčka může být zeď, skrze kterou se roboti nemohou pohybovat. Pokud má políčko na určité straně stěnu, může na ní zároveň mít také jeden až tři lasery, nebo jeden strkač (pusher). Zeď se uplatní při libovolném dění na hracím poli. Lasery a strkače, stejně jako některé typy políček působí ve hře jen v určitý čas. Každé políčko může navíc obsahovat jednu z vlajek umístěnou v hracím plánu, kterou mají hráči za úkol se svým robotem projít.

2.3 Karty

2.3.1 Karty pohybu

Hra obsahuje dva druhy karet. Prvním jsou karty pohybů pro roboty. Tyto karty představují instrukce posunu nebo rotace pro roboty. Na začátku každého kola dostane každý hráč několik karet pohybu a ty vloží do jednotlivých nezamčených registrů (viz dále) svého robota, čímž jej naprogramuje. Každá karta pohybu představuje jeden z následujících druhů instrukcí:

1) Move 1 – Pohyb vpřed o jedno políčko.
2) Move 2 – Pohyb vpřed o dvě políčka.
3) Move 3 – Pohyb vpřed o tři políčka.
4) Back up – Pohyb o jedno políčko vzad.
5) Rotate rigth – Otočení robota o 90° vpravo.
6) Rotate left – Otočení robota o 90° vlevo.

7) U-turn – Otočení robota o 180°.

Každá karta má navíc určenou prioritu, ta se použije pří stanovení pořadí provádění instrukcí roboty.

2.3.2 Karty s vylepšením

Kartu s vylepšením může hráč získat tím, že na konci kola jeho robot stojí na políčku vylepšující opravny. Každá z karet představuje nějaké vylepšení robota (např.: Double-barreled laser – zdvojení laseru, Radio kontrol – robot zasažený mým laserem bude do konce kola vykonávat program mého robota, Brakes – kdykoliv robot provádí kartu Move 1, nemusí se pohnout). Většina vylepšení upravuje chování robota jen v případě, že toto vylepšení hráč, který jej ovládá, použije. Hra má standardně definováno 26 různých vylepšení, která ovlivňují různé aspekty hry.

V případě počítačové implementace hry, která by umožňovala používat vylepšení stejně jako standardní desková hra, by každý hráč musel být dotázán pokaždé, kdy by měl možnost něco kartou vylepšení ovlivnit. To by hru mohlo velmi zdržovat a znepříjemňovat. Proto jsem se rozhodl karty s vylepšením v počítačové verzi hry neimplementovat.

Karty s vylepšením mají ještě jednu další vlastnost: pokaždé, když je robot mající nějaké vylepšení, zasažen laserem, může toto vylepšení obětovat a účinek laseru se tak na něj neuplatní. Toto chování jsem v programu zachoval v podobě štítů. Pokaždé, když má hráč obdržet kartu s vylepšením, dostane namísto toho jen štít. Ten může obětovat v případě, že je jeho robot zasažen laserem a účinek laseru se tak neuplatní.

2.4 Roboti

Hra obsahuje osm robotů. Ti jsou z hlediska herních možností rovnocenní, kromě toho, že vypadají různě a mají různá jména. Pro každého hráče se do hry přidá právě jeden robot. Hrající robot zabírá na hracím plánu jedno políčko a je vždy natočen konkrétním směrem. Roboti mají následující vlastnosti:

1) Životy – každý robot ve hře má kladný počet životů. Pokaždé, když robot zemře, je mu odebrán jeden život. Pokud má daný robot stále kladný počet životů, vrací se znovu do hry, jinak je ze hry vyřazen.
2) Poškození – každý robot může získat až devět poškození. Pokud získá desáté, zemře.

3) Následující vlajka – číslo vlajky, na jejíž políčko musí hráč robota přemístit. V okamžiku, kdy se toto povede, stane se následující vlajkou další vlajka v pořadí. Poté, co se robot dostane na poslední vlajku, úspěšně končí hru. Hráč, kterému se jako prvnímu podaří se svým robotem úspěšně ukončit hru, vyhrává.
4) Karty vylepšení – hráči mohou ve hře získávat karty s vylepšením pro své roboty. Tyto karty nejsou v programu implementovány, jsou nahrazeny štíty.
5) Štíty – pokud je robot zasažen laserem a má nějaké štíty, může tyto obětovat a vyhnout se tak účinku laseru, tj. získání poškození.
6) Registry – každý robot má pět registrů (očíslovaných 0, 1, 2, 3, 4), do nich hráč umisťuje karty pohybu během programování, karty se potom uplatní při provádění registrů. Sekvence registrů obsazená kartami tvoří program robota. Pokud má robot již čtyři poškození, za každé další je mu uzamčen jeden registr a to ten poslední nezamčený. Karty v zamčených registrech v nich zůstávají mezi jednotlivými koly a hráč s nimi nemůže hýbat.
7) Uložená pozice – každý robot má uloženou pozici, na kterou se vrací, pokud zemře.
8) Vypnutí – každý hráč se může během programování svého robota rozhodnout, že jej vypne pro příští kolo. Robot pak nebude příští kolo programován a nebude provádět žádné karty.

Umísťování karet do registrů se nazývá programování. Téměř všechny vlastnosti robotů jsou veřejné a mohou je tedy neustále vidět všichni hráči. Neveřejné jsou pouze karty obdržené jednotlivými hráči pro programování robotů. Karty nově umístěné do registrů se stávají veřejně viditelné v okamžiku provádění příslušného registru. Karty v uzamčených registrech jsou naopak stále viditelné.

2.5 Průběh hry

Hra probíhá v kolech. V jednom kole vždy probíhá nejprve programování robotů, pak provádění naprogramovaných registrů a na závěr úklid.

2.5.1 Programování robotů

Programování robotů provádějí všichni hráči současně. Nejprve se zamíchá balíček s kartami pohybu a každý hráč obdrží maximálně devět karet (přesněji od počtu devět je odečten počet poškození příslušného robota a výsledná hodnota přestavuje počet karet obdržených hráčem). Hráč při programování určí (z

obdržených karet) pro každý nezamčený registry svého robota jednu kartu dle svého uvážení. Vkládané karty pokládá lícovou stranou dolů tak, aby ostatní hráči neviděli příslušnou instrukci. Pak zbylé karty vrátí do balíčku, čímž ukončí programování. Pokud programuje již jen jeden hráč, spustí se časomíra odpočítávající 30 vteřin. Pokud poslední hráč nestihne svého robota naprogramovat před vypršením časomíry, prázdné registry jsou mu ze zbylých karet naprogramovány náhodně.

Po tom, co všichni hráči dokončí programování, oznámí, zda chtějí mít svého robota vypnutého příští kolo. Kdo robota pro příští kolo vypne, nebude ho příští kolo programovat a robot bude vypnutý, příští kolo pak začne úplně bez poškození (všechna poškození jsou mu na začátku příštího kola opravena).

Jelikož cílem je, aby ve výsledném programu hra probíhala na jednom počítači, musejí se zde pravidla změnit. V programu budou proto hráči programovat své roboty postupně, aby si navzájem neviděly karty. Zároveň při programování svého robota se každý rozhodne, jestli ho chce vypnout pro příští kolo, aby se hra zrychlila. Aby nikdo nemohl hru zdržovat, tak se před jejím začátkem zvolí pevně dané časové omezení na programování jednoho robota.

2.5.2 Provádění registrů

Před zahájením provádění registrů se všem robotům, kteří jsou toto kolo vypnutí, opraví všechna poškození. Pro každý z pěti registrů se potom postupně provedou následující kroky:

1) Všichni nevypnutí roboti provedou svou kartu pro aktuální registr. Pokud u dvou robotů záleží na pořadí, v jakém roboti své karty provedou, provede jako první svou kartu ten robot, jehož karta má vyšší prioritu. Pokud karta robota někam posouvá, jeho posun se provede postupně. Pokud má robot v cestě zeď, zarazí se o ni, pokud najede na bezednou jámu, zemře a toto kolo je vyřazen z hracího plánu. Jestliže má ve směru svého robota jiného robota posouvá jej taky, stejně jako sebe. Když robot vyjede mimo hrací plán, je to stejné, jako by spadl do bezedné jámy.
2) Pohnou se expresní dopravníky. Funguje stejně jako následující bod, ale aktivní jsou jen expresní dopravníky.
3) Pohnou se expresní i obyčejné dopravníky. Každý robot, který stojí na nějakém dopravníku, je posunut ve směru daného dopravníku, pokud to lze. Dopravník neposune robota nikdy skrz zeď, ani pokud mu v cestě překáží jiný robot (robot

nepřekáží, pokud je zároveň také odsunut dopravníkem). Jestliže je robot nasunut na jiný dopravník a ten zatáčí, je robot tímto dopravníkem otočen o 90°, směrem kam dopravník zatáčí.

4) Strkače zatlačí. Robot, který stojí před strkačem aktivním v tomto registru, je jím odsunut o jedno políčko. Při odsunu strkačem platí stejná pravidla jako by posun o jedno políčko daným směrem byl proveden kartou pohybu.
5) Převodovky se otočí. Robot stojící na převodovce je jí otočen o 90° ve směru jejích šipek.
6) Lasery vystřelí. Některá políčka na desce obsahují lasery. Navíc má každý robot svůj vlastní laser, se kterým střílí ze svého políčka směrem, kterým se dívá. Laser se zastaví o první zeď nebo robota, který je v jeho cestě. Pokud se zastaví o robota, dostane tento jeden zásah laserem. Všechny lasery ve hře vystřelí najednou, každý robot získá za každý zásah laseru jedno poškození. Pokud jich ve výsledku má více než devět, zemře.
7) Roboti se dotknou vlajek a opraven. Každý robot, stojící v této fázi na nějaké vlajce, nebo opravně (i vylepšující opravně) se jí dotkne, tím na ni posune svou uloženou pozici. Pokud robot stojí na vlajce, na kterou má právě za úkol dojít, může pokračovat na další vlajku. Byla-li vlajka již poslední, robot úspěšně končí hru.

Jestliže se během provádění registrů kterýkoliv robot ocitne mimo hrací plán, nebo v bezedné jámě, ihned zemře, je vyřazen z hracího plánu a je mu odebrán jeden život.

Provádění registrů bude v programu probíhat téměř stejně jako ve hře. Jediný rozdíl je, že pokud některý z hráčů zasažených laserem má štíty, může je obětovat. A bude proto ihned po střelbě lasery dotázán, zda a kolik štítů si přeje obětovat. Pro rychlost a jednoduchost budou všichni hráči dotázáni najednou, zde před sebou žádné informace skrývat nemusejí.

2.5.3 Úklid

Na závěr každého kola hry proběhne úklid. Každému robotu, který na konci kola stojí na nějaké opravně nebo vlajce a je poškozen, je opraveno jedno poškození. Robot, který stojí na vylepšující opravně, dostane navíc jeden štít. Pak jsou všechny karty v nezamčených registrech robotů vráceny do balíčku.

Následně jsou do hry vráceni roboti, kteří toto kolo zemřeli, ale zůstal jim ještě aspoň jeden život. Tito se do hry vracejí postupně v pořadí, v jakém stáli na startovních pozicích. Robot je vrácen vždy na políčko, na kterém má svou uloženou pozici. Pokud na něm již je jiný robot, hráč si musí vybrat libovolné přilehlé políčko, na kterém není ani jiný robot, ani bezedná jáma. Robot může být do hracího plánu umístěn natočený libovolným směrem. Vrácený robot začne hru s dvěma poškozeními a může si vybrat, jestli chce být ihned toto kolo vypnutý.

Během úklidu se mohou roboti, kteří byli vypnutí během právě proběhlého kola rozhodnout, zda chtějí zůstat vypnutí i další kolo.

3 Realizace systému

V souladu se stanovenými cíli práce byla vytvořena počítačová aplikace, která implementuje hru RoboRally, včetně dvou podpůrných programů. Celý systém byl vytvořen ve vývojovém prostředí Visual Studio v jazyce C# pro operační systém Windows. Je postaven na platformě .NET 4.0 a sestává z následujících programů:

1) RoboRally – dále jen RoboRally GUI. Okenní aplikace umožňující hraní hry RoboRally na PC ve více hráčích, nebo proti počítači vytvořená dle zadání práce. Uživatelská dokumentace k RoboRally GUI je v příloze A.
2) Evolution2 – konzolová aplikace, která byla vytvořena za účelem hledání vhodných nastavení parametrů umělých inteligencí (enginů) robotů. Aplikace využívá genetické algoritmy. Její bližší popis je uveden v kapitole *5 Nastavení parametrů pomocí genetických algoritmů*.
3) RoboRally_Console – konzolová aplikace pro hraní hry RoboRally mezi roboty ovládanými počítačem. Popsána je v kapitole *5 Nastavení parametrů pomocí genetických algoritmů*.

Jazyk C# byl zvolen proto, že umožňuje rychlý vývoj aplikací, nabízí také široké možnosti knihoven a různých možností usnadnění programování, včetně garbage collectoru. Na rozdíl od jiných alternativ je navíc platforma .NET přímo součástí nových systémů Windows, pro které je program vytvářen. Uživatel tak není zatěžován instalací dalších podpůrných knihoven. Při vývoji jednotlivých součástí systému byl jako dokumentace použit web MSDL Library [5] a kniha C# in Depth [4].

Programová dokumentace k těmto aplikacím je v příloze B.

3.1 Herní jádro

Herní jádro implementuje většinu funkcí nutných pro realizaci hry (tj. jednotlivých kol a fází). Herní jádro potom nezprostředkovává vlastní interakci s hráči ani s počítačovými hráči. Interakce s hráči poskytuje uživatelské rozhraní obsažené v RoboRally GUI (s počítačovými hráči i RoboRally_Console).

3.2 Uživatelské rozhraní programu RoboRally GUI

Grafické rozhraní programu RoboRally GUI poskytuje všechny potřebné ovládací prvky a vizualizaci hry RoboRally, která je implementována herním jádrem,

a přidává některé funkce navíc. Jedná se například o sestavení herního plánu, rozdání karet každému hráči a umožnění hráčům naprogramovat roboty, umožnění hráčům navrátit robota do hry v případě, že robot zemře, umožnit hráči použít štíty robota, pokud je to možné nebo omezit čas na programování robota (tento čas není omezen pro počítačové hráče, ti jsou dostatečně rychlí).

K vytvoření uživatelského rozhraní byla použita knihovna Windows Forms. Ta je přímo součástí .NET framework, díky tomu uživatel nemusí již instalovat nic dalšího. Bylo by možné použít i jiné knihovny k tomu určené, například novější knihovnu Windows Presentation Foundation. Windows Forms (verze 2.0) je ale podporována projektem Mono a díky tomu bude (snad) možné v budoucnu používat RoboRally GUI i na platformách, kde lze použít Mono.

Knihovna Windows Forms umožňuje snadné vytváření okenních aplikací. Visual Studio obsahuje i rozhraní pro interaktivní definování vizuálních prvků jednotlivých oken, které budou součástí aplikace. K sestavení uživatelského rozhraní bylo použito již existujících komponent z Windows Forms. Bylo však nutné i vytvoření nových komponent specielně pro hru. Tyto komponenty byly většinou vytvořeny jako podtřídy třídy `Control`, díky čemuž dobře zapadají do již existujícího systému komponent. Vytvořeno bylo i několik formulářů, jako podtřídy třídy `Form`.

3.3 Nevyřešené problémy

Program obsahuje jeden známý problém v uživatelském rozhraní. Ten se vyskytne, pokud uživatel programuje svého robota (viz A.4.2.1 Programování robotů), vyprší mu čas k tomu určený a zároveň uživatel přetahuje pomocí myši některou kartu, nebo mapu. Po vypršení časů se objeví dialogové okno s upozorněním, že čas vypršel. Při tom se zároveň kurzor myši změní v přesýpací hodiny, a tak nejde dialogové okno zavřít pomocí myši. Okno pak lze zavřít stiskem „Enter“ na klávesnici.

Při instalaci programu je nutné jako místo instalace zvolit složku, kde má oprávnění zapisovat uživatel, který bude aplikaci spouštět (například v „C:\Program files“ to u nových systémů (Windows Vista/7) není). Program totiž do některých podadresářů vytvořených při instalaci zapisuje data. Jiná možnost řešení problému je spouštět program.

4 Vytvoření počítačového hráče

4.1 Analýza hry

V implementované hře RoboRally mohou být někteří roboti ovládáni počítačem (tzv. počítačovým hráčem). K tomu bylo nutné implementovat algoritmy, které budou realizovat rozhodování hráče schopného programovat a řídit robota tak, aby tento robot (potažmo hráč) měl reálnou šanci nejenom úspěšně ukončit hru, ale případně i tuto hru vyhrát. Počítačového hráče bylo nutné vybavit „umělou inteligencí“ na úrovni rozumného hráče. Při zpracování tohoto tématu byla s výhodou využita kniha Artificial inteligence: A modern approach [2].

RoboRally není hra s úplnou informací. Všichni hráči vidí celý stav hry, tj. hrací pole a stavy robotů všech ostatních hráčů. Nevidí ale, jaké možnosti mají ostatní hráči při programování robotů. Hráč tedy nemůže předem určit, kudy by ostatní roboti v následujícím provádění registrů mohli jít, ani kam skutečně půjdou. Při rozhodování tedy není možné použít například algoritmus minimaxu často používaný ve hře Šachy. Žádný hráč navíc nemá vyhrávající strategii, bude-li jeden hráč dostávat stále jen karty rotace, nebude mít vůbec možnost se někam dostat a tedy ani vyhrát.

Program řídící rozhodování robota budu dále nazývat engine. Engine robota se musí rozhodovat v následujících situacích:

1) Programování robota – engine dostane karty a musí naprogramovat svého robota tak, aby se dostal co nejrychleji k následující vlajce. Přitom by se měl snažit, aby při svém pohybu nebyl ze své naplánované cesty odstrčen jiným robotem, případně by se mohl snažit strčit do jiného robota. Dále taky musí engine rozhodnout, zda bude výhodné robota pro příští kolo vypnout, čímž se robotovi smažou všechna poškození, ale během kola se nemůže hýbat.
2) Návrat robota – pokud robot zemře, je po konci kola vrácen do hry s ubraným jedním životem. Engine musí rozhodnout, na kterou z možných pozic robota umístit a jakým směrem ho natočit.
3) Použití štítů – pokud je robot zasažen lasery a má alespoň jeden štít, musí se engine rozhodnout, kolik ze svých štítů obětuje.

Základní funkce engine definuje rozhraní `IRobotEngine`. Díky němu si kdokoliv může napsat svůj vlastní engine tak, že toto rozhraní implementuje ve své vlastní třídě. Deklarace tohoto rozhraní je následující:

```
namespace RoboRally {
     public interface IRobotEngine {
          void Initialize(GameCore gc, Robot r);
          void RobotDied(GameCore gc, Robot r,
          List<RobotPosition> availablePositions);
          void ProgramRobot(GameCore gc, Robot r,
          List<MotionCard> cards);
          void LaserHits(GameCore gc, Robot r);
     }
}
```

Jednotlivé definice metod v rozhraní mají pouze následující čtyři druhy parametrů, ty mají ve všech metodách stejný význam:

1) `GameCore gc` – instance třídy `GameCore` obsahující celý stav hry, který engine může vidět (tj. neobsahuje přesnou kopii balíčku karet). S touto instancí si engine může dělat, co chce.
2) `Robot r` – odkaz na robota, kterého engine ovládá a musí mu nastavit potřebné údaje.
3) `List<RobotPosition> availablePositions` – seznam možných pozic, na které může engine svého robota umístit.
4) `List<MotionCard> cards` – seznam karet, které může engine použít k naprogramování svého robota.

Engine má vždy povinnost nastavit jen některé vlastnosti robota, ostatní může libovolně měnit, nebo ponechat jak jsou. Žádané vlastnosti jsou pak použity k nastavení robota, který se účastní aktuální hry a patří tomuto enginu.

Význam jednotlivých metod rozhraní je následující:

1) `Initialize` – metoda je volána před začátkem hry, umožňuje enginu se připravit na hru. Engine v této metodě nemusí provádět žádné nastavení.
2) `RobotDied` – metoda je volána během závěrečné uklízecí fáze každého kola, pokud robot patřící enginu během proběhlého kola zemřel a je nutné vrátit ho do hry. Engine musí robotu `r` nastavit jeho směr (vlastnost `Dir`) a pozici (vlastnost

`RobotPosition`). Možné hodnoty směrů a pozic jsou metodě předány v parametru `availablePositions`.

3) `ProgramRobots` – metoda je volána během programovací fáze kola, když má engine naprogramovat svého robota. Engine zde musí robotu `r` rozhodnout, zda bude vypnut v příštím kole (vlastnost `PowerDownNextRound`). Dále, pokud robot není toto kolo vypnutý (vlastnost `PowerDown`), musí mu nastavit všechny nezamčené registry, jednotlivé registry jsou dostupné indexací ve vlastnosti `Registers`.
4) `LaserHits` – metoda je volána během kroku střílení laserů, při provádění některého z registrů, a to pouze, pokud byl robot patřící enginu zasažen alespoň jedním laserem a přitom má alespoň jeden štít. Engine musí robotu r rozhodnout, kolik ze štítů robota si přeje použít (vlastnost `ShieldsUsed`). Počet použitých štítů je omezen počtem zásahů laserem, které robot dostal (vlastnost `LaserHits`) a počtem dostupných štítů (vlastnost `Shields`).

4.1.1 Použití štítů

Využít štíty se vyplatí téměř vždy. Při nepoužití štítů získá robot zbytečné poškození. To mu snižuje počet karet, které v příštích kolech obdrží, případně i zamkne karty v registrech. Tím pádem má engine méně možností jak robota programovat a sníží se tím i pravděpodobnost, že robota bude možné naprogramovat tak, aby šel výhodným směrem, čímž se pravděpodobně zpomalí postup robota k následující vlajce a sníží šance na výhru.

Nevyužít štíty se vyplatí zejména tehdy, když robot bude příští kolo vypnutý, tím se mu totiž všechna získaná poškození smažou a zásah laserem mu toto kolo hry již neovlivní. Použít štíty, když bude robot příští kolo vypnutý, se tedy vyplatí jen tehdy, je-li šance, že by robot během zbytku kola mohl získat tolik poškození, že by v tomto kole zemřel. Dále je vhodné štíty nepoužít, pokud je již jisté, že robot během tohoto kola zemře, pak by byly štíty použity zbytečně. Nevyužít štíty také může být výhodné, když engine ví, že bude na konci tohoto kola stát na vlajce, nebo opravně. Pak nevadí, když bude mít robot na konci kola jedno poškození, protože to mu bude opraveno. Engine `BaseRobotEngine` používá vždy všechny štíty, které může.

4.1.2 Návrat robota

Při návratu robota na herní plán je nutné vybrat, na kterou pozici (pozici a směr) z možných jej umístit. Potom je nutné rozhodnout, zda robota na následující kolo vypnout. Vypnutím se může robot zbavit dvou poškození, která má vždy po návratu do hry. Rozbor, zda robota vypnout je v sekci *4.1.4 Vypnutí robota*.

Pozici, na kterou robota umístit, lze například vybrat náhodně. Ve většině případů jsou totiž možné pozice jak robota umístit jen čtyři, jsou to čtyři různé směry na uložené pozici robota. Rozdíl mezi pozicemi je pak jen v natočení robota a jeho změna nebude mít na hru velký vliv (nemůže se pak stát, že by některá z možných pozic byla o hodně nevýhodnější než jiná). Přesto to nebude příliš výhodná strategie umísťování robota.

Určitě je lepší mít pro umístění robota alespoň nějaká pravidla. Například pokud je možné vybrat si z více políček, pravděpodobně bude lepší to, které je blíže následující vlajce a zároveň je výhodnější robota natočit směrem k následující vlajce.

Vhodné je rovněž ohodnotit jednotlivé možné pozice nějakou metrikou a pak z nich vybrat nejlepší. Pak záleží, jak dobrá je tato metrika. Toto bude asi výhodné hlavně, pokud již takovou metriku používáme jinde, třeba při rozhodování, jak naprogramovat robota. Engine `BaseRobotEngine` takto vybírá nejlepší pozici s použitím poziční metriky (definováno dále), kterou implementují jeho podtřídy.

Při výběru pozice je také možné zohlednit aktuální pozice ostatních robotů. Můžeme se například snažit, aby v cestě směrem na vlajku nestál našemu robotu jiný robot. Nevýhodou je, že mohou ještě zbývat jiní roboti, kteří se musejí vrátit do hry. Tito roboti mohou mít stejnou uloženou pozici a poté si vybírat z podobných možných návratových pozic a stejně našemu robotu zatarasit cestu.

4.1.3 Programování robota

Při programování robota je nutné vybrat z dostupných karet ty, které umístíme do nezamčených registrů a rozhodnout, zda robota pro příští kolo vypnout. Pokud je robot toto kolo vypnutý musíme jen rozhodnout, zda ho vypnout pro příští kolo. Zda robota vypnout rozebírá následující sekce *4.1.4 Vypnutí robota*.

Nejjednodušší možností by bylo naplnit volné registry kartami tak, jak je dostaneme, nebo karty do registrů náhodně vybrat. To by nejenže nebyl vhodný protihráč ani pro začátečníka, ale takto by robot neměl téměř vůbec šanci projít

vlajky a hru dokončit. Je velmi pravděpodobné, že by brzy vyčerpal všechny své životy a skončil.

K tomu, aby robot prošel všechny vlajky, potřebujeme pro výběr karet do jeho registrů alespoň nějaká pravidla. Například bychom mohli pro rozhodnutí, kterou kartu dáme do registru použít tzv. if-then pravidla (popsána v knize Aritficial Intelligence [2]), tedy pravidla typu:

```
if (Je před robotem díra.) then Nepoužívej žádnou kartu, která ho posune vpřed.
if (Robot je natočen směrem na vlajku a mám kartu, která ho posune vpřed.) then Tuto kartu použij.
```

Tato pravidla by se potom použila při vybírání karty do každého registru. If-then pravidla by šlo zapsat přímo do programu, nebo je v nějaké formě třeba číst ze souboru. Pak bychom mohli snáze tato pravidla upravovat a třeba je i nějak automaticky generovat a výsledné chování robotů porovnávat a vybrat nejlepší.

Další možností je na herním plánu postupně s robotem z jeho aktuální pozice zahrát všechny kombinace karet, které jsou k dispozici. Po odehrání každé z kombinací pak ohodnotit výslednou pozici robota a takto vybrat posloupnost karet, která získá nejlepší ohodnocení, tuto možnost implementuje třída `BaseRobotEngine`. Karet, které engine dostane je nevýše devět, ty musí umístit do nejvýše pěti registrů, nejvyšší počet kombinací, které lze z karet udělat je pak:

$$9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 = 15120$$

Na dnešních počítačích není problém všechny tyto kombinace postupně nasimulovat a ohodnotit. Následně je nutné definovat ohodnocovací funkci, díky které bude vybrána nejlepší výsledná pozice.

Pro dané karty lze vyhodnotit pohyb robota a jeho výslednou pozici různými funkcemi, tyto funkce budeme nadále zkráceně nazývat metrikami. Některé metriky lze potom sečíst a tím pozici hodnotit zároveň několika metrikami. Já probírané metriky pro přehlednost ještě dělím do následujících druhů:

1) Poziční metrika – ohodnocení výsledné pozice, do které se robot určitými kartami dostane. Tato metrika je vztažená k aktuální vlajce a ohodnocuje, jak blízko je robot k vlajce, nebo jak dlouho (kolik kol) bude z dané pozice trvat dojít k dané vlajce.

a) Manhattan metrika – Nejjednodušší je použít tzv. Manhattan vzdálenost (název je z knihy Artificial Intelligence [2]). Ta se počítá jako počet kroků, které by robot musel udělat, kdyby se mohl při každém kroku pohnout o jedno políčko libovolným směrem a zároveň by mohl procházet zdmi a nepadal by do bezedných jam. Je to tedy pouze součet rozdílů pozice vlajky a pozice robota v osách x a y. Tato metrika není příliš přesná. Navíc bychom mohli sestavit takovou desku, že použitím pouze této metriky by robot nemohl nikdy projít všechny vlajky. Na doporučených herních plánech by ale engine mohl dokončit hru i s touto metrikou.

b) Vylepšená Manhattan metrika – Vylepšením předchozí metriky je zohlednění bezedných jam a zdí. Tato metrika určuje, kolik nejméně kroků by robot musel udělat, aby došel k následující vlajce, s tím, že krok může být libovolným směrem o jedno políčko, ale robot nemůže procházet zdí ani nemůže přejít bezednou jámu. Tato metrika je mnohem přesnější než předchozí a nehrozí u ní, že by se robot zasekl na místě kvůli zdem nebo bezedným jámám. Metriku lze snadno spočítat předem pro všechna políčka jedním prohledáváním mapy do šířky od vlajky. S touto metrikou už robot může dokončit libovolnou mapu. Tuto metriku používá k určení pozičních hodnot `ManhattanDistanceEngine`.

c) MinCards metrika – Vzdálenost od vlajky je vhodné určit jako počet skutečných kroků robota (počet vykonaných registrů) k vlajce. To můžeme stanovit například tak, že si vytvoříme graf, možných přechodů mezi políčky pro všechny druhy karet. Uzly tohoto grafu budou možné unikátní pozice robota, pozice je určena jak políčkem kde robot stojí, tak směrem, kterým je natočen. Hrany budou mezi uzly, mezi kterými lze projít vykonáním některé z karet, a budou orientované. Tento graf nazveme poziční graf. Na tomto grafu lze pak počítat různé metriky. Poziční graf bychom měli vytvořit pětkrát, strkače totiž mohou být v každém z registrů aktivní nebo neaktivní a grafy se pak mohou pro jednotlivé registry lišit. Přesto o něm budu dále mluvit jako o grafu jediném a budu předpokládat, že pro každý registr bude použit odpovídající graf.

   Nejjednodušší metrikou založenou na pozičním grafu je vyjádření minimálního počtu karet, se kterými se lze dostat na pozici, kde je vlajka. Toto lze opět spočítat dopředu pro všechny pozice prohledáváním do šířky.

Robot se ale typicky nemůže dostat k následující vlajce za tolik kroků, kolik je určeno touto metrickou.

d) AverageCards metrika – Metrika spočítaná na pozičním grafu by se měla co nejvíce přiblížit skutečnému počtu karet, které budou muset být robotem provedeny, aby se dostal na následující vlajku. To znamená, že by bylo dobré spočítat průměrný počet karet, které robot provede, než se z dané pozice na vlajku dostane.

Nalezení průměrného počtu karet použitých k posunu robota z daného políčka na následující vlajku můžeme spočítat jako: $\sum_{i=1}^{\infty} i \cdot p_i$, kde p_i označuje pravděpodobnost, že se robot dostane na následující vlajku použitím i karet. p_i vypočteme jako:

$$p_i = \sum_{\substack{s \in \{(a_1,\dots,a_i) | robot\ se\ z\ dané\ pozice \\ vykonáním\ posloupnosti\ karet\ a_1,\dots,a_i \\ dostane\ na\ následující\ vlajku\}}} P(s),$$

kde karty a_1 až a_i jsou z balíčku karet hry, stejné karty se v posloupnosti mohou opakovat. P(s) značí pravděpodobnost, že hráč ovládající robota dostane a použije posloupnost karet s.

Výpočet P(s) si zjednodušíme a odhadneme ho jako pravděpodobnost, že při provádění tahů s vracením z balíčku karet postupně vytáhneme posloupnost s. Díky tomu jsou si všechny karty stejného typu pro následující výpočet ekvivalentní, tedy:

$$P(s) = P(a_1, \dots, a_i) \cong \sum_{j=1}^{i} r(a_j) = \sum_{j=1}^{i} \frac{počet\ karet\ typu\ a_j\ v\ balíčku}{počet\ všech\ karet\ v\ balíčku},$$

kde $r(a_j)$ je pravděpodobnost, že z balíčku karet vytáhneme kartu stejného typu jako je a_j (provádíme tahy s vracením). Tímto jsme sice zhoršili přesnost výpočtu průměrného počtu karet, ale výpočet je díky tomu také významně jednodušší. Přestože řada $\sum_{i=1}^{\infty} i \cdot p_i$ má nekonečně mnoho členů, výpočet provedeme jen pro prvních n členů, tím se opět zhorší přesnost výpočtu, ale pro dostatečné n již ne tak významně jako při předchozím zjednodušení. Vzorec pro výslednou metriku je tedy následující:

$$\sum_{i=1}^{n} i \cdot \Big(\sum_{\substack{s \in \{(a_1,\dots,a_i) | robot\ se\ z\ dané\ pozice \\ vykonáním\ posloupnosti\ karet\ a_1,\dots,a_i \\ dostane\ na\ následující\ vlajku\}}} \sum_{j=1}^{i} \frac{počet\ karet\ typu\ a_j\ v\ balíčku}{počet\ všech\ karet\ v\ balíčku} \Big)$$

K výpočtu metriky nazvané AverageCards použijeme dynamické programování. Postupně budeme u všech pozic najednou počítat, s jakou pravděpodobností, se z dané pozice dostaneme na tu následující vlajku na 0, 1, 2,… kroků. K tomuto účelu si pravděpodobnosti pro 0 až n kroků budeme u každého uzlu pozičního grafu pamatovat. P[i] bude pro daný vrchol grafu označovat vypočtenou hodnotu pravděpodobnosti pro i karet. Na začátku nastavíme všechna P[i] všech vrcholů na 0, pouze u čtyř vrcholů představujících políčko s vlajkou nastavíme P[0] na 1. Tímto máme v P[0] u všech vrcholů nastaveno, s jakou pravděpodobností se z nich na 0 karet dostaneme na vlajku.

Pak provedeme cyklus od 1 do n-1, i bude označovat jeho řídící proměnnou. V těle cyklu postupně projdeme všechny vrcholy, pro každý vrchol v projdeme všechny hrany (v, w) vycházející z v a vypočteme:

$$P_v[i] = \sum_{\substack{\{(v,w)|(v,w)\ je\ hrana\\ vycházející\ z\ v,\\ kterou\ robot\ projde\ použitím\ karty\ a\\ z\ pozice\ odpovídající\ vrcholu\ v\}}} P_w[i-1] \cdot r(a)$$

Tím pro všechny vrcholy vypočteme, s jakou pravděpodobností se z nich dostaneme na vlajku pomocí i karet. Jednotlivé P_w[i-1] jsou již z minulého průběhu cyklu vypočtené pravděpodobnosti, že se na vlajku dostaneme pomocí i-1 karet, všech okolních vrcholů. Po proběhnutí tohoto cyklu vypočteme pro jednotlivé vrcholy v: $\sum_{i=0}^{n} i \cdot P_v$[i], čímž dostaneme žádanou metriku AverageCards.

Podobně lze pro každou pozici vypočítat, kolika průměrně lasery projdeme, když se z ní vydáme na vlajku. Metriku AverageCards se zjištěním počtu laserů používá `DynaProgEngine`.

2) Pohybová metrika – ohodnocení kudy robot prochází. Hodnocení vůči nebezpečím, ke kterým se přiblíží, nebo s ohledem na ostatní roboty.
   a) Zohlednění bezedných jam – pokud to není nutné, určitě není dobré držet se v blízkosti bezedných jam (nebo okrajů hracího plánu), pokud někde blízko je jiný robot. Ten by mohl našeho robota do jámy shodit. Naopak bychom se mohli preferovat cestu, která by měla šanci jiného robota shodit do jámy, to by šlo lehce u robotů, kteří jsou právě vypnuti. U nevypnutých robotů bychom museli odhadnout, kudy se budou pohybovat, což by šlo jen těžko.

b) Zohlednění ostatních robotů – které karty použijeme, můžeme plánovat úplně bez ohledu na ostatní roboty. Je ale lépe, snažit se ostatním robotů vyhnout tak, aby nás nemohli vychýlit z námi naplánované trasy. Nevíme ale, kam přesně ostatní roboti půjdou. Mohli bychom například pouze snížit body za to, že se přiblížíme k místu, kde se jiný robot nachází, před prováděním karet. Robot se ale, jen provedením jednoho registru může posunout až o pět políček. Proto je vhodnější použít k vyhledání možných budoucích výskytů robota poziční graf. Na něm můžeme rychle nalézt všechny jeho možné výskyty po provedení jednotlivých registrů. Zároveň můžeme podobně jako u metriky AverageCards odhadovat s jakou pravděpodobností robot danou hranu pozičního grafu použije, odhadneme ji jako pravděpodobnost, že si kartu odpovídající dané hrané vytáhne z balíčku.

   Lepší, než zjišťovat, na kterém políčku se bude moci daný robot nacházet, je zjistit, přes která všechna políčka se v tom kterém registru mohl pohybovat při provádění své karty. Pro zpřesnění můžeme očekávat (ale to se vůbec nemusí stát), že se daný robot bude pohybovat na políčka, která zlepší jeho pozici vůči jeho následující vlajce, a těmto pohybům dát větší pravděpodobnost. Tuto možnost implementuje `BaseRobotEngine`.

c) Zohlednění laserů robotů – zohlednit můžeme také to, kam by mohli ostatní roboti při provádění registrů střílet svými lasery. K tomu potřebujeme odhadnout, kam se asi budou pohybovat, což rozebírá předchozí bod. Když už odhadujeme, kudy se robot bude pohybovat, dáme u toho zároveň záporné body políčkům, na která by mohl střílet svým laserem. Tuto možnost implementuje `BaseRobotEngine`.

Ohodnocení výsledné pozice robota musí také zahrnovat kladné body za to, že robot došel na další vlajku, nebo získal štít a záporné body za získaná poškození.

4.1.4 Vypnutí robota

O tom, zda robota vypnout, musí engine rozhodovat po naprogramování robota a při jeho návratu do hry poté, co zemřel. Rozhodnout se musí, zda robota vypnout pro následující kolo. V obou těchto případech víme, kde robot příští kolo bude. Při vracení robota do hry bude přesně na místě, kam ho umístíme. Po naprogramování robota sice nevíme jistě, jestli robot dojde, kam jsme ho chtěli dostat, může jej

během cesty ovlivnit jiný robot a změnit mu trasu, ale budeme předpokládat, že se toto nestalo.

Víme (předpokládáme) tedy, v jakém stavu robot bude. Teď musíme rozhodnout, zda bude v tomto stavu vhodné, aby se vypnul. Nejjednodušší způsob by mohl být vypínat robota náhodně s nějakou pravděpodobností, pokud bychom vůbec nevěděli, podle čeho se rozhodovat. Rozhodovat bychom se ale určitě měli podle toho, jak moc je robot poškozen. Robota pak můžeme vypínat při získání nějakého pevného počtu poškození, nebo s počtem získaných poškození zvyšovat pravděpodobnost, že ho vypneme.

Při rozhodování zda robota vypnout bychom měli vzít do úvahy, kam by se robot mohl příští kolo dostat. Pokud robotu nehrozí v příštím kole smrt, díky velkému počtu poškození, může i s několika zamčenými registry jít směrem, který se nám bude hodit. Navíc vypnutím robota, pokud by například stál na páse, který jej po vypnutí odveze do bezedné jámy, určitě není výhodné. Proto bychom se měli snažit odhadnout, kam se robot dostane, pokud by se pro příští kolo vypnul a pokud ne, možné pozice ohodnotit a podle toho se rozhodnout. Zjistit, kam se robot dostane, pokud je vypnutý, je snadné, vypnutého robota posunují pouze dopravníky. Nevypnutý robot se může dostat do různých pozic, podle toho jaké obdrží karty, můžeme se ale snažit získat průměr pozic, do kterých se robot dostane. Ten získáme průchodem pozičního grafu a zprůměrováním možných pozic. Takovéto zprůměrování pozic pak určitě poskytne horší pozici, než do jaké se skutečně dostaneme. Z toho důvodu bude vhodné robota vypnout, až když bude rozdíl mezi vypočtenými hodnotami dostatečný. Tento postup implementuje `BaseRobotEngine`.

4.2 Realizace

Jako základ enginu, byla vytvořena třída `BaseRobotEngine`. Tato třída implementuje všechny metody rozhraní `IRobotInterface` (toto rozhraní musejí enginy implementovat), sama ale přidává další abstraktní metodu, tou je:

```
protected abstract void InitiatePathCosts(int flagInd,
GameCore gc);
```

Tuto metodu musí třída odvozená `BaseRobotEngine` implementovat. V ní je nutné inicializovat pole `positionValues[flagInd]` v němž jsou uloženy tzv.

poziční metriky jednotlivých políček vůči vlajce, informace o dané vlajce (číslo a pozice) jsou uloženy v `flags[flagInd]`. Pro každou unikátní pozici robota na hracím plánu (tj. umístění a natočení robota) pole `positionValues[flagInd]` obsahuje hodnotu této pozice vůči dané vlajce. Tato hodnota je pevná po celou dobu hry a vyjadřuje, jak výhodná je tato pozice vůči příslušné vlajce (vyšší hodnota představuje výhodnější pozici). Engine tato data využívá při hledání nejlepšího naprogramování robota. Při inicializaci enginu je tato metoda volána pro každou vlajku na hracím plánu.

Třída `BaseRobotEngine` implementuje algoritmy rozhodování, její podtřídy ale musejí nastavit poziční metriky všech pozic vůči vlajkám na hracím plánu. Byly vytvořeny dvě její podtřídy, třída `ManhattanDistanceEngine` používající Vylepšenou Manhattan metriku a třída `DynaProgEngine` používající AverageCards metriku. `BaseRobotEngine` používá při svém rozhodování metody popsané v předchozích podkapitolách (v předchozích podkapitolách je použitých metod nasáno, které třídy je používají).

Při rozhodování na kterou z možných pozic robota vrátit je nejlepší z možných pozic vybrána pouze pomocí poziční metriky jednotlivých pozic. Při programování robota jsou postupně procházeny všechny možnosti dané kartami, které engine dostal, na pozičním grafu. Hodnoceny jsou výsledné pozice a cesty, kterými se na ně robot dostane, vybrána je pozice s nejvyšším ohodnocením. Zohledňovány jsou možné pozice a lasery ostatních robotů po provedení jednotlivých registrů. Ty jsou určeny prohledáním všech možných pozic ostatních robotů pomocí pozičního grafu. Závěrečné pozice, do kterých se robot dostane, jsou ohodnoceny pomocí poziční metriky, počtu prošlých vlajek, získaných poškození a štítů. Při rozhodování o vypnutí robota je ohodnocena pozice, do které se robot vypnutím dostane, a jsou zprůměrovány hodnoty pozic, do kterých by se robot mohl dostat, kdyby nebyl vypnut. Robot je vypnut, pokud hodnota jeho pozice po kole kdy by byl vypnutý, dostatečně převyšuje průměr hodnot, do kterých by se mohl dostat nevypnutý. Pokud je možnost použít štíty, je vždy použito maximum možných štítů.

Následující podkapitoly blíže popisují implementaci jednotlivých metod rozhraní `IRobotEngine` třídou `BaseRobotEngine` a implementaci metody `InitiatePathCosts` jejími podtřídami. Při svých rozhodováních (v těchto metodách) používá `BaseRobotEngine` a její podtřídy různé parametry, které jsou

zde rovněž popsány. Přiřazení vhodných hodnot těmto parametrům je potom uvedeno v kapitole *5 Nastavení parametrů pomocí genetických algoritmů*.

4.2.1 Metoda Initialize

V této metodě je provedena inicializace dat, která se v průběhu hry nemění. Do pole `cardProbs` jsou uloženy pravděpodobnosti jednotlivých druhů karet. Do seznamu vlajek `flags` jsou uloženy všechny vlajky na herním plánu. Pole `damageCosts` je naplněno hodnotami pro jednotlivá poškození, které se vypočtou z parametrů `dtCostA` a `dtCostB`, pro n poškození je hodnota vypočtena jako:

$$n^2 \cdot dtCostA + n \cdot dtCostB$$

Pak je vytvořen graf všech možných pozic robota na mapě (poziční graf). Ten má jeden uzel pro každé unikátní umístění a natočení robota. Z každého jeho uzlu pak vede sedm hran (tolik je ve hře různých typů karet pohybu), vždy do uzlu odpovídajícímu pozici, do níž by se robot danou kartou dostal, pokud by byl na hracím plánu sám. Pro každou z hran, které z uzlu vycházejí, je při vytváření grafu navíc uloženo, přes která políčka se robot při posunu do nového uzlu pohybuje a s jakou šancí robot tuto hranu použije, pokud bude stát „na jejím začátku“. Tato šance pro jednotlivé hrany vedoucí z políčka je vždy menší než jedna a pro každou hranu je přímo úměrná pravděpodobnosti, že hráč dostane kartu, odpovídající hraně a tomu o kolik si použitím této hrany zlepší poziční metriku. (To se pak využije ke zjišťování, zda se vybraná cesta bude pravděpodobně křížit s cestou vybranou jiným robotem.) K tomu je v každém uzlu speciální hrana, která určuje, kam by se robot z daného políčka dostal, po proběhnutí jednoho registru, kdyby byl vypnutý. Tento graf je uložen v proměnné `graph`. Do proměnné `reversedGraph`, je pak uložena transpozice tohoto grafu, už bez dodatečných informací. Následně je pro všechny vlajky na herním plánu zavolána metoda `InitializePathCosts`.

4.2.2 Metoda RobotDied

Nejlepší pozice, na kterou robota umístit se vybírá jejich ohodnocením pouze pomocí poziční metriky, uložené v `positionValues`. Nic jiného se nezohledňuje. Rozhodnutí o vypnutí robota probíhá stejně jako v metodě `ProgramRobot` viz dále.

4.2.3 Metoda ProgramRobot

Vybírání nejlepší kombinace karet probíhá prověřením všech možností a jejich ohodnocením. Nejprve je však pro jednotlivé registry vypočteno, kterými políčky a s jakou pravděpodobností by mohli ostatní roboti při provádění těchto registrů procházet, a na která políčka by mohli střílet. Vypočtené pravděpodobnosti, kde by mohli ostatní roboti být, jsou vynásobeny parametrem `robotCost`, pravděpodobnosti pro střelbu jsou vynásobeny parametrem `robotLaserCost`, a součet těchto dvou pro jednotlivá políčka je pak uložen do pole `registerPositionVals`. Základní ohodnocení pozice se s pomocí parametrů `flagVal`, `shieldValue` a hodnot v `damageCosts` vypočte jako:

$$(počet\ vlajek, které\ robot\ prošel) \cdot flagVal + (počet\ štítů\ robota) \cdot shielValue + damageCosts[počet\ poškození\ robota]$$

K tomu je ještě přičtena hodnota z `positionValues` odpovídající pozici, na kterou se robot s pomocí daných karet dostane. V průběhu prohledávání pozičního grafu jsou za jednotlivé pozice, které robot projde, přičítány hodnoty z pole `registerPositionVals`. Tím dostaneme celou hodnotu pozice.

Po naprogramování robota je nutné dále rozhodnout, jestli ho pro příští kolo vypneme. Do proměnné `pdVal` si vypočteme hodnotu pozice, do které by se robot vypnutím dostal. Výpočet provedeme stejně jako při hledání nejlepších karet k naprogramování robota, ale již nezohledňujeme možné pozice ostatních robotů. Pak si do proměnné `possibleVal` vypočteme průměr hodnot všech pozic, do kterých by se robot mohl příští kolo dostat. Hodnotu jednotlivých pozic vypočteme stejně jako při výpočtu `pdVal`. Následně si do proměnné `diff` uložíme rozdíl: $pvVal - possibleVal$. A s využitím parametrů `powerDownCoefA` a `powerDownCoefB` otestujeme podmínku:

$$diff \cdot powerDownCoefA + powerDownCoefB > 0$$

Pokud je tato podmínka splněna, robota vypneme.

4.2.4 Metoda LaserHits

Metoda slouží pro stanovení počtu použitých štítů. Engine vždy použije maximum štítů, které může.

4.2.5 Metoda InitializePathCosts

V této metodě musejí podtřídy `BaseRobotEngine` nastavit poziční metriku pro vlajku danou parametrem `flagInd`, který představuje index do pole `positionValues`. Od třídy `BaseRobotEngine` jsou odvozené třídy `ManhattanDistanceEngine` a `DynaProgEngine`, které tuto metodu implementují.

Třída `ManhattanDistanceEngine` k tomu používá Vylepšenou Manhattan metriku a parametr `stepCost`. Od dané vlajky prochází herní plán prohledáváním do šířky, které zohledňuje zdi a bezedné jámy, v každé další úrovni prohledávání je poziční hodnota pozic v této úrovni horší o `stepCost`. Navíc pozice (políčko a natočení robota), která směřuje na pozici s lepším ohodnocením než má sama, je zvýhodněna o -`stepCost`/2.

Třída `DynaProgEngine` používá metriku AverageCards, která pro každou pozici odhaduje průměrný počet karet, kterými se na ní stojící robot dostane na následující vlajku. Zároveň také počítá průměrný počet laserů, který robot získá při cestě k následující vlajce. K výpočtu poziční hodnoty políčka jsou používány parametry `stepCost` a `initLaserCost`:

$$(\textit{průměrný počet karet k následující vlajce}) \cdot \textit{stepCost} + (\textit{průměrný počet získaných laser}\text{ů}) \cdot \textit{initLaserCost}$$

5 Nastavení parametrů pomocí genetických algoritmů

5.1 Možnosti nastavení parametrů enginů

Každý z vytvořených enginů používá při rozhodování více než deset číselných parametrů, a to nejenom celočíselné ale i s plovoucí desetinnou čárkou. Jednotlivé parametry tedy mohou nabývat mnoha různých hodnot.

Pro nalezení optimálního nastavení enginu robota by bylo nutné všechna tato nastavení vyzkoušet. Navíc nemáme žádnou objektivní funkci, která by vyjádřila, jak je engine s daným nastavením parametrů „dobrý“. Porovnání enginů lze provést pouze hraním her proti sobě. Hra může probíhat na různých herních plánech, v různých počtech robotů a hráči dostávají každé kolo náhodné karty. Porovnání dvou enginů je tedy pomalé, jedna hra trvá řádově jednotky až desítky sekund podle počtu hrajících robotů, velikosti hracího plánu a podle toho, jak jsou hrající enginy dobré. Pro omezení vlivu náhody pro účely vzájemného porovnání enginů v různých podmínkách je nutné zahrát her více. Z výše uvedených důvodů je zřejmé, že není reálné zkoušet všechna možná nastavení enginů a provádět jejich vzájemné porovnávání, to by zabralo příliš mnoho času. Je tedy nutné zvolit jinou strategii, která nám umožní vyhledat optimální respektive vhodné nastavení enginu v dostatečně krátkém čase.

Kniha Artificial Intelligence [2] uvádí jako jedno z možných řešení při prohledávání velkých stavových prostorů algoritmy lokálního prohledávání. Tyto algoritmy neprohledávají stavový prostor (prostor možných nastavení enginů, obecně prostor nějakých uzlů) systematicky. Proto si udržují v paměti vždy jen jeden nebo několik uzlů (nastavení enginů) a není pro ně důležité, jak jsme se k danému uzlu (nastavení) dostali. Tyto vlastnosti nám nevadí, není důležité, jak se k danému nastavení enginu dostaneme, ale chceme najít to nejlepší (nebo alespoň nějaké dobré). V [2] jsou uvedený následující možné algoritmy:

1) Metoda největšího stoupání (Hill-climbing)
2) Simulované žíhání (Simulated annealing)
3) Lokální paprskové prohledávání (Local beam search)
4) Genetické algoritmy (Genetic algorithms)

Já jsem si jako metodu pro nastavení parametrů robotů vybral genetické algoritmy. Ty nepotřebují pevnou ohodnocovací funkci, stačí jim umět porovnávat různá nastavení (různě nastavené enginy).

5.2 Genetické algoritmy

Genetické algoritmy se při hledání co nejlepšího řešení inspirují tím, jak probíhá „hledání“ nejlepších jedinců v přírodě. Rovněž terminologie používaná v souvislosti s genetickými algoritmy je obdobná terminologii používané v přírodních vědách, viz například An Introduction to genetic algoritmhs [3]:

a) Populace – skupina jedinců.
b) Jedinec (chromosom) – kandidát na řešení problému. Skládá se z jednotlivých genů, které kódují různé jeho vlastnosti.
c) Selekce – vybrání jedinců v populaci k reprodukci. Čím lepší jedinec, tím častěji bude vybrán k reprodukci.
d) Křížení – zkombinování dvou jedinců k vytvoření nového jedince.
e) Mutace – změna náhodná změna genetického kódu jedince.

Základní genetický algoritmus pak vypadá následovně (schéma algoritmu bylo převzato z An Introduction to genetic algorithms [3]):

1) Začneme s náhodně vygenerovanou populací n m-bitových jedinců (kandidátů na řešení problému).
2) Vypočteme ohodnocení (fitness) každého jedince v populaci.
3) Opakujeme následující, dokud nevytvoříme n potomků:
   a) Vybereme dva rodičovské jedince z aktuální populace, pravděpodobnost vybrání jedince jako rodiče je rostoucí funkcí jeho ohodnocení. Výběr provádíme „s vracením“, jedinec tedy může být jako rodič vybrán vícekrát.
   b) S pravděpodobností p_c (pravděpodobnost křížení) zkřížíme pár vybraných rodičů k vytvoření dvou potomků. Pokud křížení neprovedeme, dva noví potomci se stanou přesnými kopiemi svých rodičů.
   c) Zmutujeme potomky v každém jejich místě (bitu) s pravděpodobností p_m (pravděpodobnost mutace) a výsledné jedince umístíme do nové populace.
4) Nahradíme aktuální populaci novou populací.
5) Jdeme do kroku 2.

Jedna iterace tohoto procesu se nazývá generace, průběh více generací se pak nazývá běh.

5.3 Realizace

5.3.1 Reprezentace jedinců

Implementované enginy mají každý buďto dvanáct (`ManhattanDistanceEngine`), nebo třináct (`DynaProgEngine`) číselných parametrů, které v genetickém algoritmu představují jednotlivé geny jedince. Tyto parametry si enginy umějí načíst z textového souboru, kde je na každém řádku jeden parametr. Pro zjednodušení jsou v souboru všechny tyto parametry reprezentovány čísly v rozsahu 0 až 65536. Engine si tento rozsah pak musí převést do rozsahu hodnot, který je vhodný pro jeho parametry (po provedení běhů pro enginy se tento systém zdá být dostačující). Díky tomu je i usnadněna práce programu, který celý běh řídí, ten může se všemi geny pracovat stejně. V každém souboru je uloženo patnáct parametrů (hodnot genů), i když nejsou všechny použité. Počet je dostačující pro oba implementované enginy a je tak možné se soubory pro oba typy enginů pracovat stejně.

5.3.2 Ohodnocení jedinců

Jelikož nemáme žádnou objektivní ohodnocovací funkci, jsou jedinci v populaci ohodnocování souboji s jinými jedinci v populaci. Souboje se konají vždy ve dvou, čtyřech, nebo osmi. Aby bylo zajištěno, že každý jedinec může získat stejné ohodnocení, probíhají souboje v kolech. V daném kole se každý robot vždy zúčastní jedné hry. Podle toho v kolika hráčích se hraje, jsou vždy postupně vybírány dvojice, čtveřice, nebo osmice robotů, kteří v tomto kole ještě nehráli, a vybraní si proti sobě zahrají. Z tohoto důvodů musí být velikost populace vždy násobkem osmi.

Souboje jedinců probíhají úplně stejně jako normální hra, jen se jich účastní pouze enginy a každý souboj má časové omezení. Omezení času na hru je nutné, protože někteří jedinci by nestihli dokončit hru, nebo zabít svého robota v rozumném čase. Každá hra se hraje na náhodně zvoleném hracím plánu z těch, které jsou ve hře předpřipravené. Roboti enginů jsou na prvních n (kde n je počet hrajících robotů) startovních pozic rozmístěni náhodně. Při hrách v různých počtech jsou enginy za dosažené pozice ohodnoceni různě. Následující popisuje, jak jsou jedinci za jednotlivé výsledné pozice ohodnoceni (více bodů znamená lepší ohodnocení, zleva jsou vedle sebe napsány body od první pozice dále):

1) Hry ve dvou: 4, 0

2) Hry ve čtyřech: 8, 4, 1, 0
3) Hry v osmi: 16, 8, 4, 2, 1, 1, 0, 0

Jedinci, kteří hru vůbec nedokončí, buď proto, že zemřou, nebo proto, že již vypršel časový limit, nezískají žádné body.

5.3.3 Porovnání populací

Abychom mohli zhodnotit, jak se populace vyvíjejí, musíme je umět porovnávat. Tolik nás ale nezajímá, jak se vyvíjí populace jako celek, jako to, jak se vyvíjejí její nejlepší jedinci. Proto mezi sebou porovnávám vždy čtyři nejlepší jedince populací. Mohli bychom porovnávat jen nejlepšího jedince, ovšem, díky způsobu jeho výběru, tento nemusí být v populaci nejlepší. Se čtyřmi jedinci z každé populace (při vzájemném porovnávání dvou populací) dostaneme osm jedinců, což je maximální počet možných hráčů ve hře. S těmito jedinci pak sehrajeme zápasy ve dvou, čtyřech a osmi stejně jako při ohodnocování jedince uvnitř populace (popsáno v předchozí podkapitole 5.3.2 Ohodnocení jedinců), s tím, že do každé hry je vždy vybrán stejný počet jedinců z obou skupin nejlepších jedinců. Ohodnocení jedinců v každé skupině jsou potom sečtena, tento součet udává ohodnocení dané populace.

5.3.4 Selekce

K výběru jedinců, kteří se budou křížit a dají vznik nové populaci, je použita úprava tzv. selekce pomocí ruletového kola (roulette-wheel selection, popsáno v [3]). Při této selekci je šance jedince být vybrán přímo úměrná jeho ohodnocení. Použitý algoritmus je velmi podobný, šance výběru jedince je ale přímo úměrná tomu, o kolik je lepší než nejhorší jedinec. To platí pro všechny jedince kromě toho nejhoršího, ten má taky šanci být vybrán, ale jen velmi malou. Z každé populace do další automaticky přechází nastavený počet nejlepších jedinců.

5.3.5 Křížení a mutace

Vybraní jedinci jsou buďto přesunuti do nové populace ihned, nebo se provede jejich křížení a mutace. Při vytváření nového jednice křížením do něj z jeho předků kopíruji vždy celé geny (odpovídající parametrům enginu). V případě křížení jsou vždy provedena dvě nezávislá zkřížení, čímž vzniknou dva potomci postupující do nové populace. Způsob křížení je vždy vybrán náhodně z následujících:

1) Jednobodové křížení – náhodně vygenerujeme číslo n, menší než je celkový počet genů jedince. Prvních n genů bude nový potomek mít od prvního rodiče, zbytek od druhého rodiče (pořadí rodičů je náhodné).
2) Dvoubodové křížení – náhodně vygenerujeme čísla m a n, jejichž součet je menší, než počet genů jedince. Prvních n genů dostane potomek od prvního rodiče, dalších m od druhého rodiče a zbytek opět od prvního rodiče (pořadí rodičů je náhodné).
3) Stejnoměrné křížení – u každého z genů potomka náhodně zvolím, jestli jej dostane od prvního, nebo druhého rodiče.
4) Průměrování – každý z genů potomka je aritmetickým průměrem odpovídajících genů jeho rodičů.

U nového potomka každý gen zmutuji s nastavenou pravděpodobností. Při mutaci genu jeho hodnotu změním o číslo vygenerované z normálního rozdělení se střední hodnotou nula a nastaveným rozptylem. Tím pádem je pravděpodobnější menší změna genu, velké změny jsou mutací způsobeny jen málokdy.

5.4 Aplikace RoboRally_Console

RoboRally_Console je konzolová aplikace umožňující provádět souboje mezi roboty ovládanými počítačem. Možné parametry této aplikace jsou následující:

1) *–m* <*mapa*> – volba hracího plánu, na kterém bude hra probíhat. <*mapa*> musí být název jednoho z hracích plánů, které jsou uloženy ve složce „./maps“. Tento hrací plán musí mít rozmístěné všechny startovní pozice a alespoň jednu vlajku. Tento parametr je povinný.
2) *–l* <*cesta*> – <*cesta*> udává jméno souboru, do kterého bude ukládán záznam průběhu hry. Pokud tento parametr není zadán, záznam není nikam ukládán.
3) *–r* <*startovní pozice*> <*typ enginu*> <*cesta*> – Touto volbou je do hry přidán jeden robot. Tento robot bude hru začínat na startovní pozici číslo <*startovní pozice*>. Engine, který bude robota řídit, je určen parametrem <*typ enginu*>, „M“ znamená `ManhattanDistanceEngine` a „D“ `DynaProgEngine`. <*cesta*> určuje cestu k souboru s parametry pro tento engine. Hry se musí vždy zúčastnit alespoň dva roboti a nejvýše osm, proto musí být tato volba zadána alespoň dvakrát a nejvýše osmkrát.

Výstupem aplikace je pouze výsledek hry. Nejprve jsou vypsáni roboti, kteří hru dokončili, v pořadí jak skončili. Za každého robota, který hru dokončil, je na jeden

řádek vypsáno číslo jeho startovní pozice. Startovní pozice jsou unikátní a lze tak pomocí nich roboty určit. Pokud některý z robotů hru nedokončil, následuje za výpisem těch, kteří hru dokončili prázdný řádek a následně výpis robotů, kteří zemřeli (na každém řádku číslo startovní pozice jednoho robota, který zemřel).

5.5 Aplikace Evolution2

Evolution2 je konzolová aplikace realizující hledání vhodných parametrů zvoleného enginu pomocí genetických algoritmů. Použití genetických algoritmů je implementováno tak, jak popisuje celá tato kapitola.

Aplikace může být spuštěna ve dvou režimech. V normálním režimu aplikace stále dokola vytváří nové populace a ohodnocuje je, kvůli vytvoření dalších populací. Každá vytvořená populace má své číslo určené pořadím, ve kterém byla vytvořena, číslování začíná od nuly. Ohodnocené populace jsou porovnány s předchozí populací a s populací, jejíž číslo je nejbližším nižším násobkem třiceti. Díky tomu můžeme během vývoje běhu vidět srovnání mezi populacemi.

Výsledky běhu aplikace jsou ukládány do zvolené složky. Soubory a složky, o kterých je psáno dále, jsou vždy vytvářeny v této složce. Pro informace o každé vytvořené populaci je, vytvořena nová složka, její název je číslo dané populace. V této složce je uloženo následující:

1) Soubory, jejichž jméno je číslem. Tyto soubory jsou uložená nastavení odpovídající jedincům populace. Formát tohoto souboru je popsán v *5.3.1 Reprezentace jedinců*.
2) Soubory „best00“ až „best03“, jsou uložení čtyři nejlepší jedinci populace, pomocí těchto jedinců jsou pak populace porovnávány.
3) Soubor „log.txt“ obsahující záznam průběhu ohodnocování jedinců dané populace. Tento soubor obsahuje postupně: informace o nastavení běhu, výpis všech jedinců populace, výpis všech provedených her, výsledná získaná hodnocení všech jedinců populace a záznam vytváření nové populace.
4) Soubor „fitnesses.txt“ obsahující výpis ohodnocení všech jedinců populace. Každý řádek tohoto souboru obsahuje vždy „číslo jedince: ohodnocení“.
5) Soubor „bestfitnesses.txt“ obsahující ohodnocení čtyř nejlepších jedinců ve stejném formátu jako „fitnesses.txt“.
6) Soubory „fitness_<*X*>vs<*Y*>.txt“ obsahující ohodnocení nejlepších čtyř jedinců populací <*X*> a <*Y*> z jejich porovnávání. Soubor obsahuje dva stejné záznamy,

nejprve pro populaci <*X*> a pak <*Y*>. Záznam je tvořen výpisem součtu ohodnocení jedinců, kteří se souboje za danou populaci účastnili, a pak výpisem ohodnocení jednotlivých jedinců ve formátu stejném jako v „fitnesses.txt“.

7) Soubory „log_<*X*>vs<*Y*>.txt“ obsahuje záznam porovnání populací <*X*> a <*Y*>. Tento soubor obsahuje postupně: Výpis jedinců, kteří se porovnání účastní za jednotlivé populace, výpis všech provedených her, výpis ohodnocení získaných jednotlivými jedinci.

Druhým režimem, v němž může být aplikace spuštěna, je porovnávací režim. V tomto režimu dojde pouze k porovnání dvou zvolených populací a výsledek je uložen do souborů „log_<*X*>vs<*Y*>.txt“ a „fitness_<*X*>vs<*Y*>.txt“.

Aplikace má následující parametry příkazové řádky:

1) *–p <číslo>* – určuje číslo populace, která má být načtena, pokud chceme pokračovat v přerušeném běhu.
2) *–m <desetinné číslo>* – nastavení pravděpodobnosti mutace (p_m).
3) *–c <desetinné číslo>* – nastavení pravděpodobnosti křížení vybraných jedinců (p_c).
4) *–t <číslo>* – počet vláken, která budou použita při hraní her.
5) *–e <číslo>* – počet nejlepších jedinců, kteří beze změny přecházejí do nové populace.
6) *–s <číslo>* – velikost populace, musí být násobek osmi.
7) *–d <cesta>* – adresář, kam budou ukládány a odkud budou načítány jednotlivé populace a záznamy.
8) *–M <číslo>* – nastavení odmocniny rozptylu náhodné veličiny z normálního rozdělení, která určuje velikost mutace při mutaci genů.
9) *–E <písmeno>* – určuje použitý engine. „M“ znamená `ManhattanDistanceEngine` a „D“ znamená `DynaProgEngine`.
10) *–R <číslo>* – nastavení omezení počtu populací, které budou vygenerovány.
11) *–T <X> <Y>* – nastavení omezení času pro hrané hry. Hra s n hráči bude omezena na $< X > \cdot n + < Y >$ milisekund.
12) *–P <cesta>* – nastavuje cestu k programu RoboRally_Console.
13) *compare <X> <Y>* – spustí program v porovnávacím módu a pouze porovná populace <*X*> a <*Y*>. Bez tohoto parametru bude program spuštěn v normálním módu.

14) *fitnessGames* <*X*> <*Y*> <*Z*> – určuje, kolik bude při ohodnocování jedinců populace hráno kol ve hrách se dvěma (<*X*>), čtyřmi (<*Y*>) a osmi hráči (<*Z*>).
15) *compareGames* <*X*> <*Y*> <*Z*> – určuje, kolik bude při porovnávání populací hráno kol ve hrách se dvěma (<*X*>), čtyřmi (<*Y*>) a osmi hráči (<*Z*>).

(Formát zadávaných desetinných čísel je dán nastavením systému, na kterém program spouštíme. Například na systému, kde byly experimenty prováděny, nešla zadat čísla ve formátu 3.14 ani 3,14, číslo muselo být zadáno ve formátu 314e-2.)

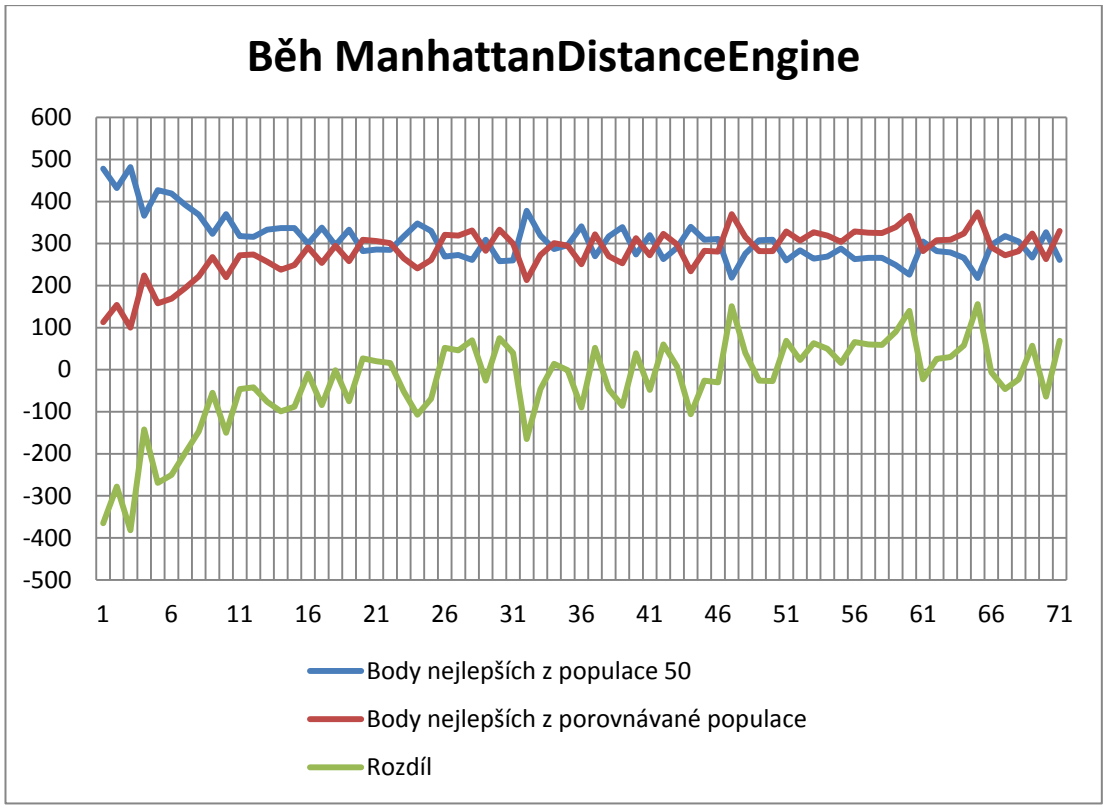
5.6 Výsledky

5.6.1 Generování první populace

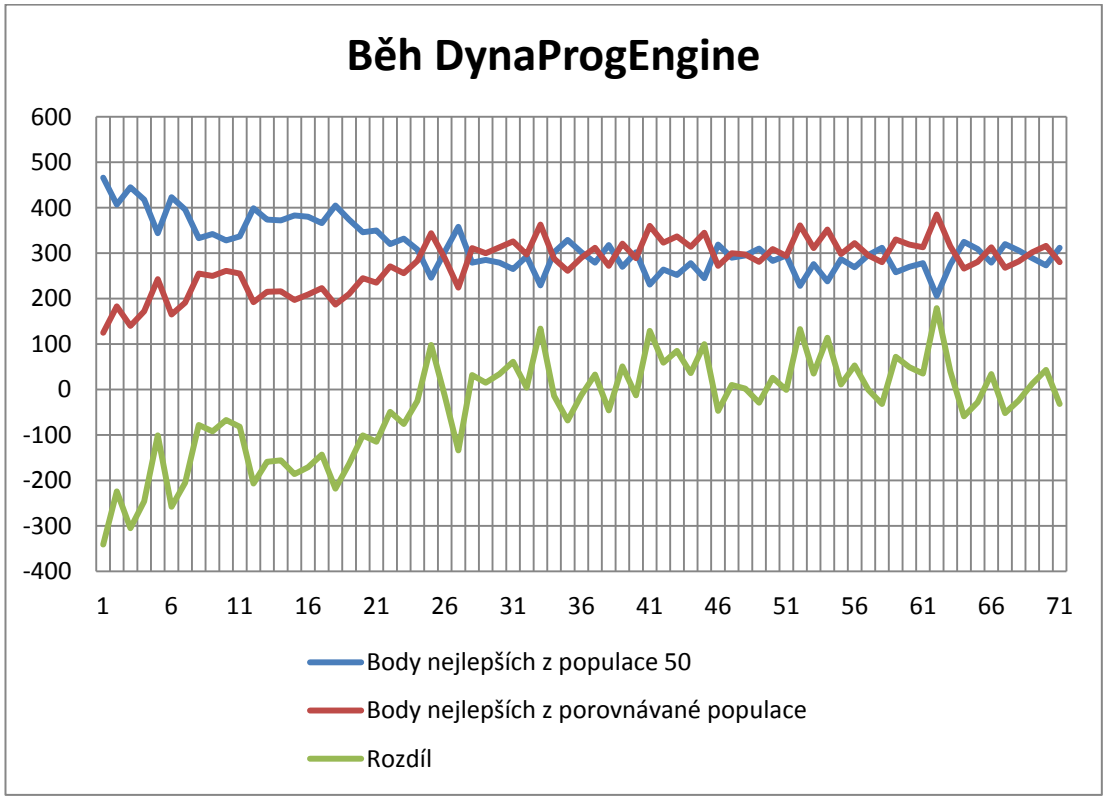
První populace by měla být generována náhodně. Náhodným vygenerováním populace ale dostaneme jen málo jedinců (typicky řádově méně, než jaká je velikost populace), kteří jsou schopni projít všechny vlajky. Pak by ale následující populace, byla z velké části tvořena pouze potomky několika málo jedinců, což není vhodné. Pro dosažení většího počtu schopných jedinců na začátku můžeme nejprve spustit běh s řádově větším počtem jedinců, než kolik v populaci chceme. Tento běh spustíme pouze s minimálním porovnáváním jedinců mezi sebou, ale s velkým počtem elitních jedinců, kteří přejdou do nové populace automaticky. Běh necháme běžet pouze do vytvoření jedné nové populace, pokud jsme nastavili, že n nejlepších jedinců jde do nové populace automaticky, jsou jedinci nula až n-1 nové populace nejlepší jedinci z předchozí populace. Při dostatečně velké populaci většina z elitních jedinců získá nějaké body. Takto dostaneme menší populaci schopných jedinců, se kterými můžeme začít nový běh.

5.6.2 Běhy

Byly vykonány dva hlavní běhy, jeden s `ManhattanDistanceEngine` a jeden s `DynaProgEngine`.



Obrázek 1 Graf porovnávající jednotlivé populace s populací číslo 50 v běhu vytvořeném k enginu ManhattanDistanceEngine, k porovnání populací bylo použito vždy osm kol her ve dvou, čtyřech i osmi hráčích (běh, ze kterého byl tento graf vytvořen je na přiloženém CD ve složce „experimenty\manhattan“)



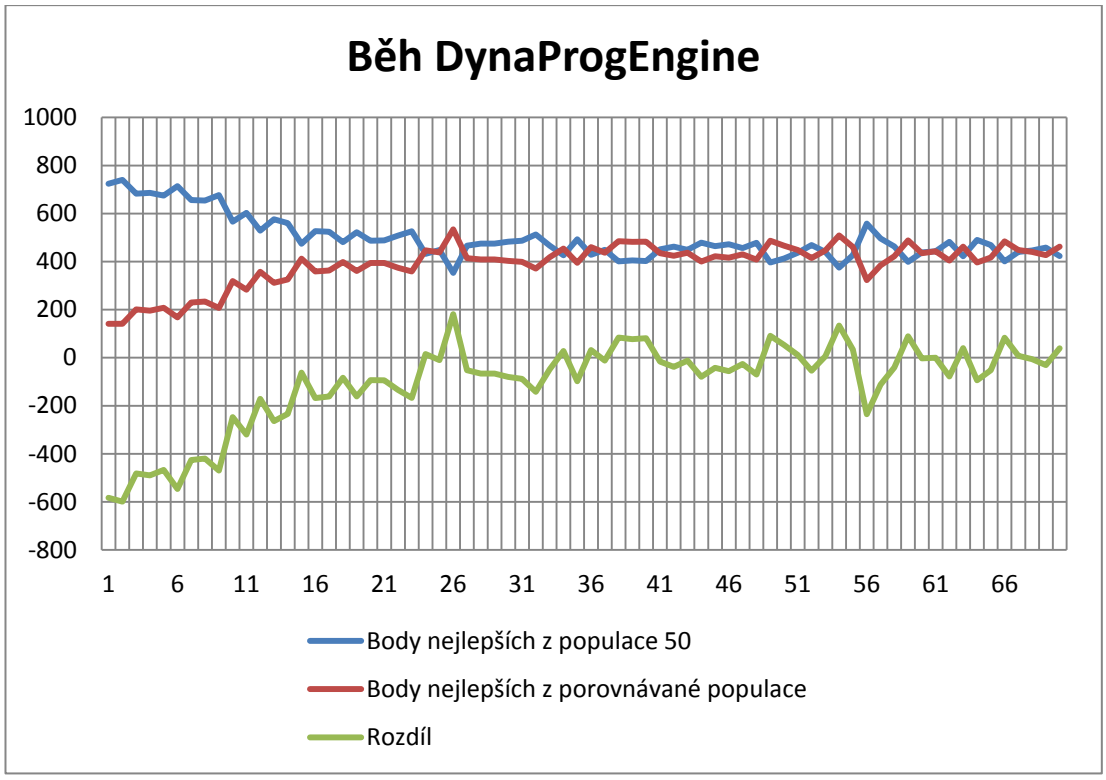
Obrázek 2 Graf porovnávající jednotlivé populace s populací číslo 50 v běhu vytvořeném k enginu DynaProgEngine, k porovnání populací bylo použito vždy osm kol her ve dvou, čtyřech i osmi hráčích (běh, ze kterého byl tento graf vytvořen je na přiloženém CD ve složce „experimenty\dynaprog1“)

Grafy na obrazcích 1 a 2 popisují běhy s enginem `ManhattanDistanceEngine` a `DynaProgEngine` respektive. Aby byl vidět postupný vývoj populací, obsahují grafy srovnání 71 populací s populací číslo 50 (číslo populace udává pořadí, ve kterém byla vygenerována). Jednotlivé křivky stejných barev mají v obou grafech stejný význam. Na ose x grafů jsou čísla populací, se kterým je populace číslo 50 porovnávána (způsob porovnání dvou populací je popsán v *5.3.3 Porovnání populací*). Na ose y jsou ohodnocení, případně rozdíl ohodnocení z jednotlivých porovnání populací.

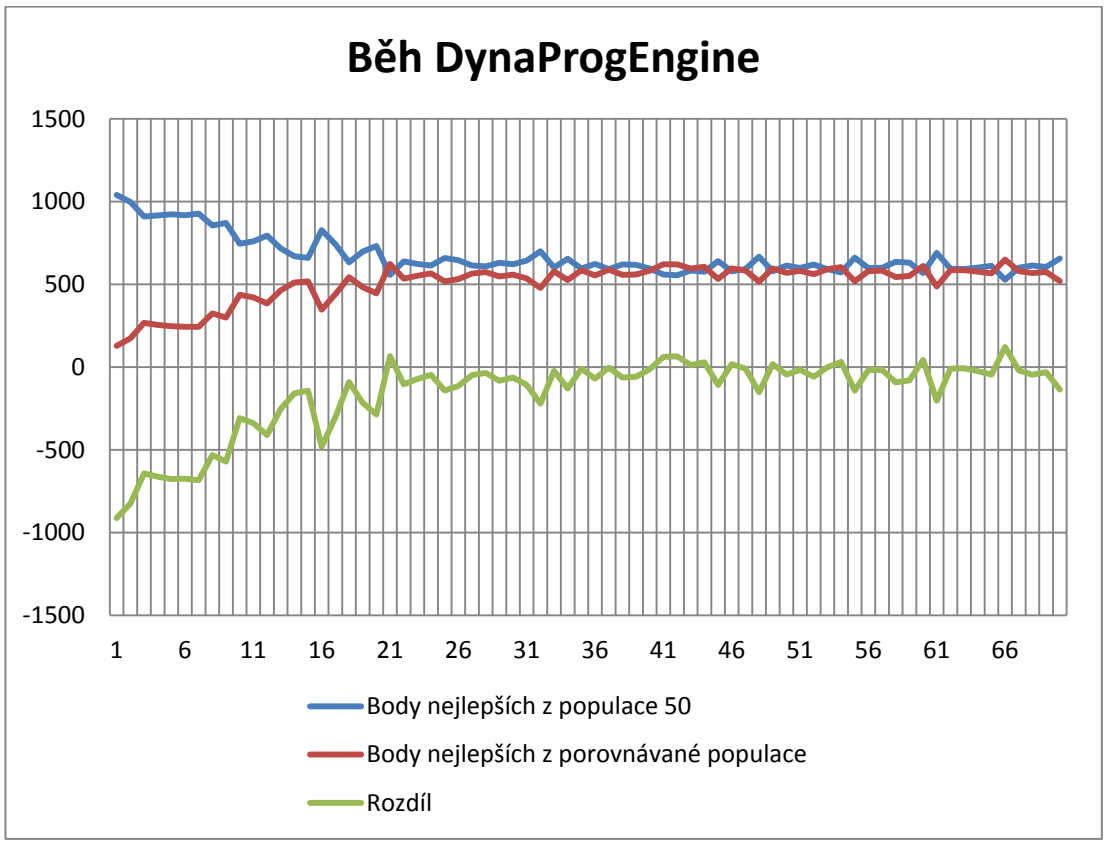
V bodě n na ose x je vynesen výsledek porovnání populace n a populace 50. Pro každé z porovnání jsou vyneseny tři body, každý patřící do jiné křivky:

1) Modrou barvou je vyznačeno, jaké ohodnocení v provedeném porovnání získala populace číslo 50.
2) Červenou barvou je vyznačeno, jaké ohodnocení v provedeném porovnání získala populace číslo n.
3) Zelenou barvou je vyznačen rozdíl mezi hodnoceními získanými populací číslo 50 a populací číslo n. Pokud je rozdíl menší než nula, znamená to, že byla lepší populace číslo 50, jinak byla lepší populace číslo n.

Na obou grafech je vidět podobný průběh. Jak se populace vyvíjejí je nejlépe vidět na zelené křivce. Počátek zelené křivky až přibližně do populace číslo 20 pro obrázek 1 a do populace číslo 25 pro obrázek 2 má rostoucí trend. To jasně ukazuje, že se zde nastavení eginů stále zlepšují. Další vývoj už ale nemá jasný rostoucí trend, křivka osciluje kolem nuly. Populace se zde již nelepší. Vychýlení křivky v některých místech bude pravděpodobně způsobeno nepřesností v porovnávání populací, hra probíhá na náhodně zvolených mapách a v každé hře hraje velkou roli náhoda.



Obrázek 3 Graf porovnávající jednotlivé populace s populací číslo 50 v běhu vytvořeném k enginu DynaProgEngine, k porovnání populací bylo použito vždy dvanáct kol her ve dvou, čtyřech i osmi hráčích (běh, ze kterého byl tento graf vytvořen je na přiloženém CD ve složce „experimenty\dynaprog2“)



Obrázek 4 Graf porovnávající jednotlivé populace s populací číslo 50 v běhu vytvořeném k enginu DynaProgEngine, k porovnání populací bylo použito vždy šestnáct kol her ve dvou, čtyřech i osmi hráčích (běh, ze kterého byl tento graf vytvořen je na přiloženém CD ve složce „experimenty\dynaprog2“)

Na obrázcích 3 a 4 je další běh s `DynaProgEngine`. Tento běh byl spuštěn s téměř stejným nastavením jako předchozí běh s `DynaProgEngine`, ale s populacemi velikosti 240 jedinců. Přesto vývoj populace proběhl prakticky stejně, jako v předchozím případě. Rozdíl mezi těmito dvěma grafy je různý počet her při porovnávání populací. Pro tvorbu prvního grafu (obrázek 3) bylo při porovnávání dvou populací sehráno dvanáct kol her ve dvou, čtyřech a osmi hráčích. U druhého grafu bylo pro všechny typy her sehráno šestnáct kol. Na grafech je vidět, že od dvacáté populace se již populace nezlepšují a že čím více her použijeme k porovnávání populací, tím více je zbytek grafu vyhlazen tedy, že výkyvy na grafu na obrázku 3, i na předchozích, jsou opravdu způsobeny především faktorem náhody.

6 Srovnání s existujícími implementacemi

Na internetu lze najít několik jiných implementací hry RoboRally pro PC. Nalezené implementace umožňují hraní RoboRally „po síti“, vytváření vlastních desek a, až na jednu výjimku, neobsahují karty s vylepšeními. Většina programů je dostupná zdarma.

1) gametableonline.com – Na této stránce je možné spustit si demo hry, které umožňuje hrát na jediné mapě proti jednomu protihráči ovládanému počítačem. Po registraci (která je zdarma) by hraní proti počítači mělo být dostupné na libovolných herních plánech a mělo by mít možné i sestavit si vlastní hrací desky. Hra proti jiným lidským hráčům po internetu je dostupná za poplatek. Po registraci se mi však hru nepodařilo spustit, takže funkce nemám ověřené. Tato hra má zajímavější grafickou stránku a obsahuje i karty s vylepšeními. Pro hru je nutné mít připojení k internetu. Program je multiplatformní (napsán v jazyce Java).
2) roborally.clandestine.dk – Neumožňuje hru proti počítačovým hráčům. Program lze spustit pouze v operačním systému Windows.
3) irristor.net/index.php?id=roborally – Umožňuje hru proti počítačovým hráčům. Program se mi povedlo spustit a vytvářet v něm hrací plány, ale samotnou hru se mi začít nepodařilo. Program je multiplatformní (napsán v jazyce Java).
4) rr6.sourceforge.net – Verze RoboRally pro unixové systémy. Neumožňuje hru proti počítačovým hráčům.

Všechny tyto implementace potřebují, aby každý hrající člověk měl vlastní počítač, a aby tyto počítače byly síťově propojené. Žádná z nich (podle toho co jsem měl možnost vidět) neumožňuje ukládání a načítání rozehraných her. Jinak jsou jejich možnosti velmi podobné.

7 Závěr

V průběhu realizace této bakalářské práce byl vytvořen počítačový program implementující deskovou hru RoboRally. Program má grafické uživatelské rozhraní a umožňuje hru jednoho nebo více hráčů na jednom počítači (s operačním systémem Windows). Do hry je možné přidat i počítačem řízené hráče. Až na popsané výjimky jsou dodržena pravidla a možnosti ze stolní předlohy. Program umožňuje sestavení hracího plánu z předem připravených desek. Vytváření vlastních hracích desek je možné pouze jejich „ručním“ napsáním do textového souboru.

Pro realizaci umělé inteligence řídící počítačové hráče byly vytvořeny dva enginy s různými způsoby rozhodování. Oba enginy byly podrobeny testování a lepší z nich (`DynaProgEngine`) byl nakonec použit ve výsledné verzi programu. Jelikož je rozhodování vytvořených enginů závislé na různých parametrech, byl vytvořen konzolový program Evolution2 hledající vhodné nastavení těchto parametrů pomocí techniky genetických algoritmů. Dále byl vytvořen program RoboRally_Console, umožňující provádět souboje mezi různě nastavenými počítačovými hráči, který je používán programem Evolution2. Pomocí Evolution2 byla nalezena rozumná nastavení jednotlivých enginů. Nejlepší nastavení, nalezené při jednom z běhů `DynaProgEngine`, bylo použito pro nastavení enginu používaného v RoboRally GUI. Enginy jsou s nalezenými nastaveními člověku dobrými protihráči a při hře jsou schopni průměrného lidského hráče porazit (vývoj hry velmi závisí na tom, jaké kdo dostane karty).

Realizací uvedených programů RoboRally GUI, RoboRally_Console a Evolution2 tak byly naplněny stanovené cíle práce. Na základě testování a zkušebního provozu se potom ukazuje, že by bylo vhodné doplnit program RoboRally GUI doplnit o nápovědy hráčům, čímž by bylo usnadněno začínání s programem především hráčům, kteří neznají pravidla RoboRally. Bylo by také vhodné vytvořit grafickou aplikaci usnadňující uživatelům vytváření vlastních hracích desek, ty je nyní nutné ručně psát do textových souborů. Rovněž by bylo dobré dále experimentovat s programem Evolution2 a různými nastaveními jeho běhů, nebo například přidáním jiných způsobů selekce. Program RoboRally_Console by bylo možné rozšířit tak, aby umožňoval nejen použití vestavěných enginů, ale i enginů, které by byly dodány jako již zkompilované knihovny. Tak by bylo možné snadno pořádat souboje mezi novými herními strategiemi. To by také vytvořilo nový

způsob hry mezi lidskými hráči, každý hráč by mohl vytvořit svůj vlastní engine a až mezi těmito enginy by se souboje odehrávaly. O tutéž vlastnost by se dala rozšířit i aplikace RoboRally GIU, pak by bylo možné souboje různých enginů přímo sledovat.

Seznam použité literatury

1) WIZARDS OF THE COAST LLC. *RoboRally rulebook*. 2005, 33 s.
2) RUSSEL, Stuart a Peter NORVIG. *Artificial Intelligence*: *A Modern Approach*. Second edition. Upper Saddle River, New Jersey: Pearson Education, Inc., 2003. ISBN 0-13-080302-2.
3) MITCHELL, Melanie. *An Introduction to Genetic Algorithms*. First MIT Press paperback edition. Cambridge, Massachusetts: A Bradford Book The MIT Press, 1998. ISBN 0−262−13316−4 (HB), 0−262−63185−7 (PB).
4) SKEET, Jon. *C# in Depth*. Greenwich: Manning Publications Co., 2008. ISBN 1933988363.
5) Dokumentace k .NET Framework 4. MICROSOFT. *MSDN Library* [online]. [cit. 2012-07-23]. Dostupné z: http://msdn.microsoft.com/en-us/library/w0x726c2

Příloha A: Uživatelská dokumentace

Tento text obsahuje uživatelskou dokumentaci programu RoboRally, který implementuje stejnojmennou deskovou hru. V implementaci pro PC je navíc možné, aby roli libovolného hráče zastával počítač. Program ke svému běhu potřebuje operační systém Windows (Windows XP a novější) a .NET framework 4.0.

A.1 Základní pravidla hry

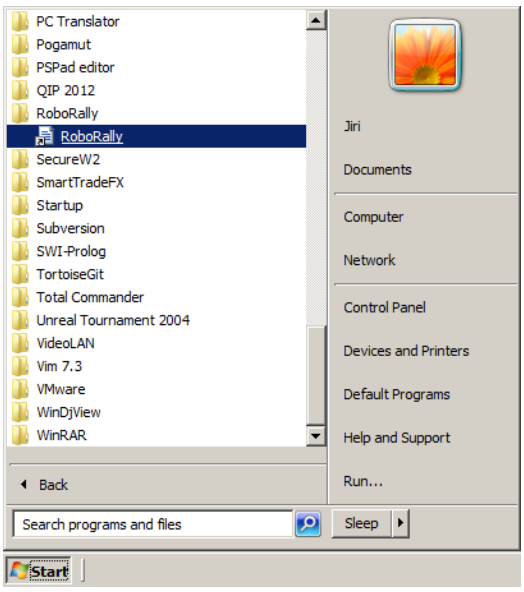
Před zahájením hry hráči nejprve sestaví z desek hrací plán (mapu) a rozmístí na něj jednu až osm vlajek. Poté může začít vlastní hra. Každý hráč ve hře ovládá jednoho vybraného robota, který začíná na některé ze startovních pozic. Úkolem je projít všechny rozmístěné vlajky ve správném pořadí. Hra probíhá v jednotlivých kolech. V každém kole nejprve každý hráč obdrží karty v maximálním počtu devět a následně všichni hráči naprogramují svého robota, tzn. umístí své vybrané karty do nezamčených registrů svého robota. Hráč v tomto okamžiku navíc může určit, že jeho robot bude příští kolo vypnutý.

Poté, co každý hráč naprogramuje svého robota, se postupně provede všech pět registrů všech robotů. To proběhne v pěti fázích, pro každý registr jedna. V každé fázi nejprve všichni roboti provedou příslušný pohyb, který jim určuje karta pro daný registr, a potom se postupně pohnou všechny prvky, které jsou na desce v pořadí: expresní dopravníky, normální i expresní dopravníky, převodovky, strkače. Na závěr každé fáze ještě vystřelí lasery, za každý zásah laserem získá robot poškození. V dalších kolech dostane hráč o tolik karet méně než devět, kolik má jeho robot poškození. Po provedení všech pěti fází se pokračuje dalším kolem.

Cílem každého hráče je postupně se svým robotem projít všechny vlajky, které jsou rozmístěné v hracím plánu, a to v pořadí jejich čísel od nejmenšího. Komu se to povede jako prvnímu, vyhrává.

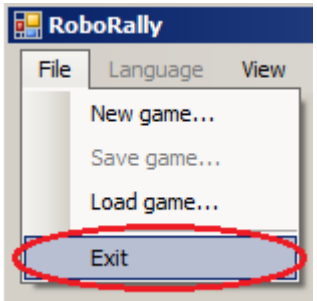
A.2 Spuštění a vypnutí aplikace

Aplikaci spustíme tak, že v menu *Start* otevřeme *Všechny programy*, zde najdeme složku *RoboRally* a v ní odkaz na program jménem *RoboRally*. Tímto odkazem (klinutím na něj) program spustíme.

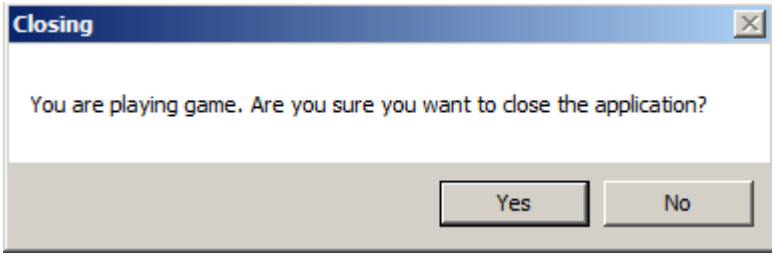


Obrázek 5 Odkaz na program RoboRally v menu Start

Aplikaci lze vypnout kliknutím na křížek v pravé části záhlaví okna, nebo vybráním položky *Exit* v menu *File* hlavního okna. Pokud byste právě měli rozehranou hru, budete dotázáni, zda chcete aplikaci opravdu vypnout.



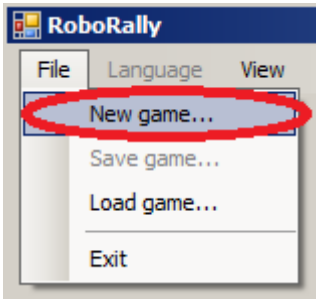
Obrázek 6 Položka Exit v menu File



Obrázek 7 Dialog s dotazem

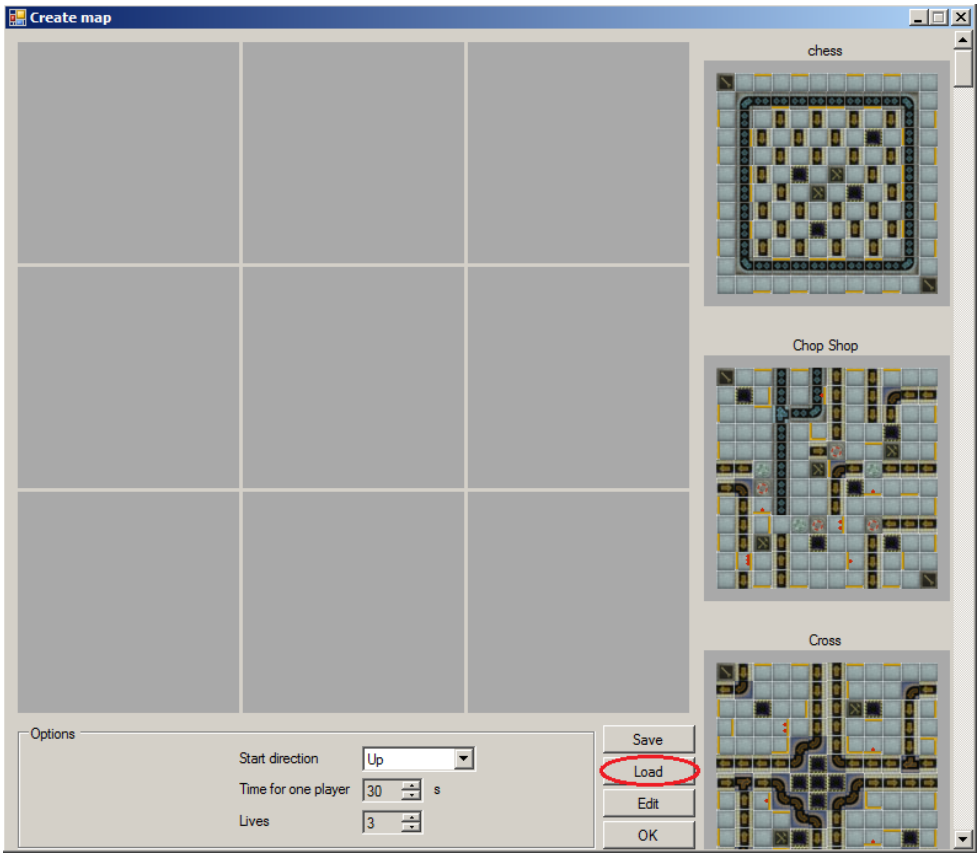
A.3 Jak začít hrát

Sestavování nové hry začněte výběrem položky *New game* v menu *File* hlavního okna.



Obrázek 8 Položka New Game v menu File

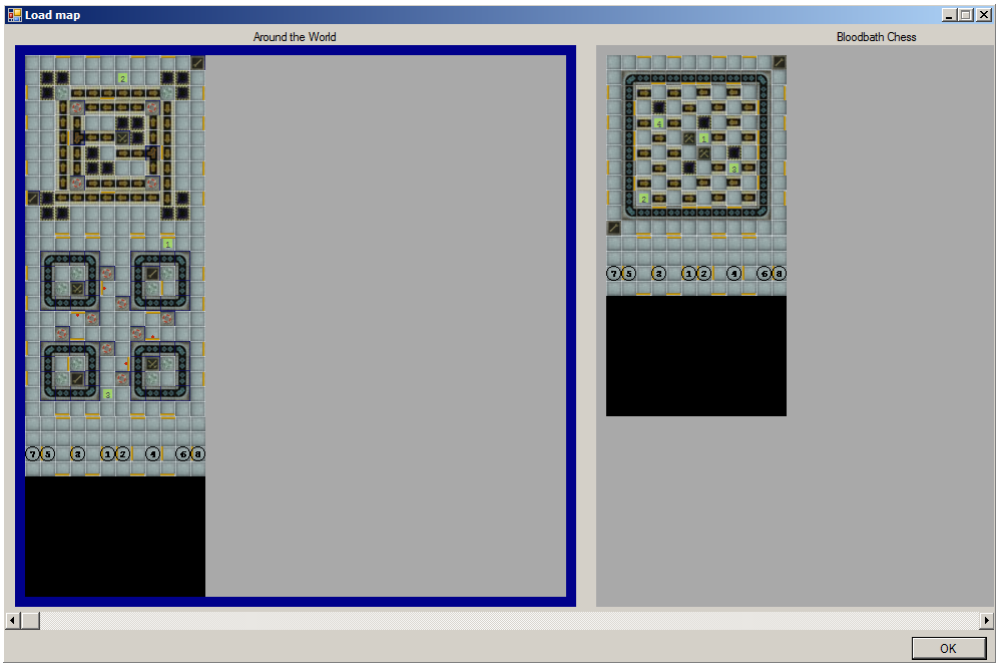
Po výběru této položky menu se zobrazí okno *Create map*. To slouží k sestavení mapy, na které bude hra probíhat.



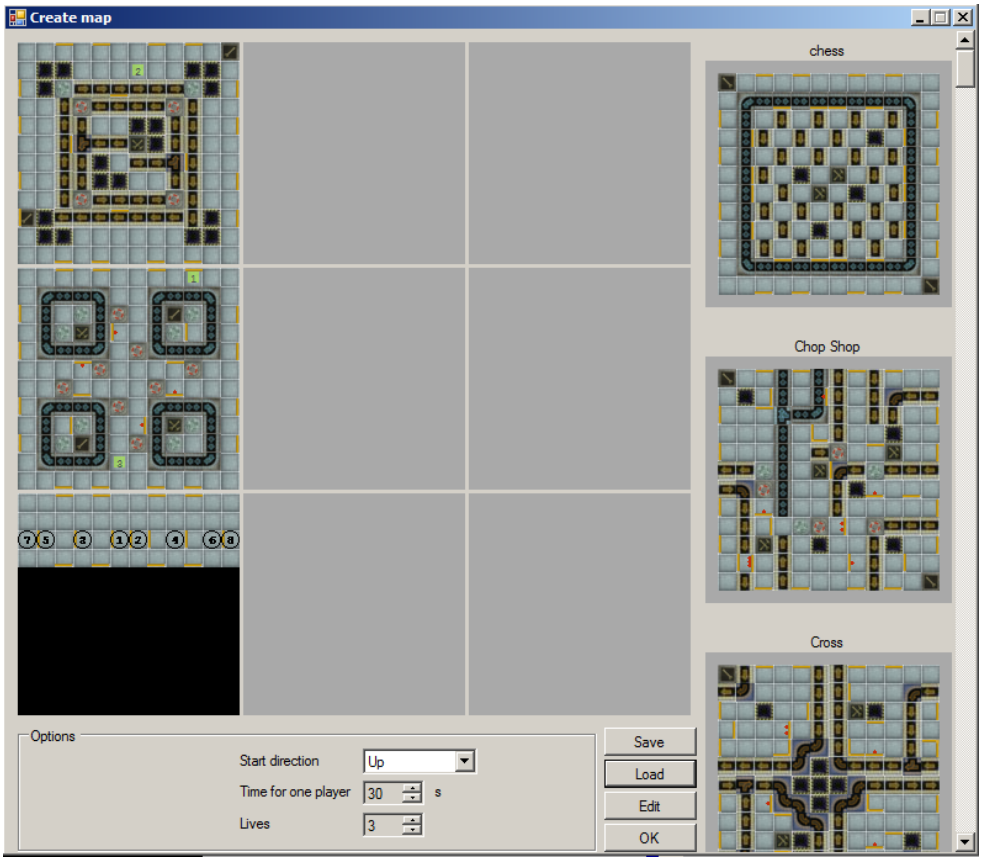
Obrázek 9 Okno Create map

V okně *Create map* pokračujte stiskem tlačítka *Load*. Tím se otevře okno *Load map* s možností výběru předpřipravených herních map. Těch hra ihned po instalaci obsahuje jedenáct a další je možné přidávat. V tomto okně vyberte některou předpřipravenou mapu kliknutím na její obrázek, vybraná mapa je vždy zvýrazněna

modrým rámečkem okolo. Stiskem tlačítka *OK* se vybraná mapa načte do okna *Create map* a okno *Load map* se zavře.

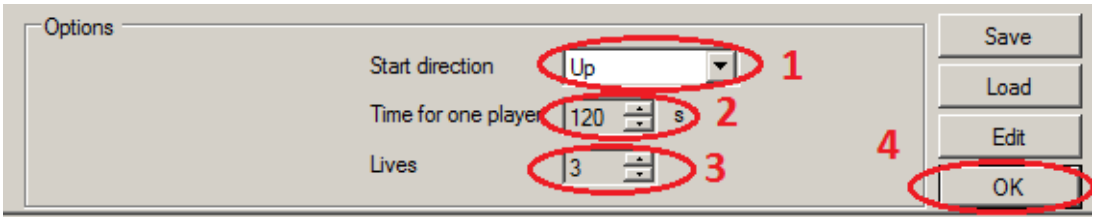


Obrázek 10 Okno Load map



Obrázek 11 Okno Create map po načtení uložené mapy

Nyní v okně *Create map* nastavte možnosti hry podle vašeho uvážení.

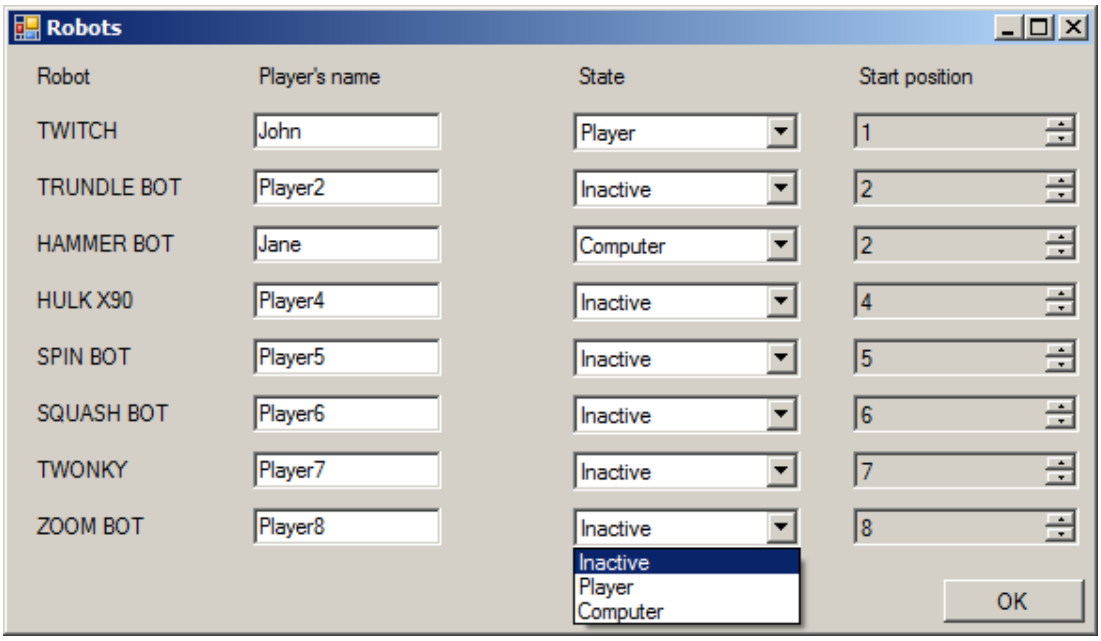


Obrázek 12 Nastavení hry

Nastavit lze následující:

1) Směr, kterým budou všichni roboti po startu hry natočeni.
2) Čas, který má každý hráč na sestavení programu svého robota. Číslo 0 znamená, že čas na sestavení programu robota není omezen.
3) Počet životů, který každý robot bude mít při zahájení hry.

Po volbě nastavení pokračujte stiskem tlačítka *OK* okna *Create map*. Zobrazí se okno s nastavením hrajících robotů.



Obrázek 13 Nastavení hrajících robotů

V okně *Robots* nastavte, kteří roboti se budou účastnit hry. Každý robot má pevně přiřazené jméno a obrázek, podle kterého si ho jeho vlastník pozná na mapě. Všichni roboti jsou až na jméno a obrázek stejní (tj. z pohledu herních možností rovnocenní). Pro každého robota, který se bude účastnit hry, nastavte následující:

1) *Player`s name* – Jméno hráče, který bude robota ovládat. Každý robot ve hře, i ten, který je ovládán počítačem, musí mít unikátní jméno hráče.

2) *State* – Stav robota, má tři možné varianty:

   a) *Inactive* – Daný robot se hry neúčastní.

   b) *Player* – Robot se bude účastnit hry a bude ovládán člověkem.

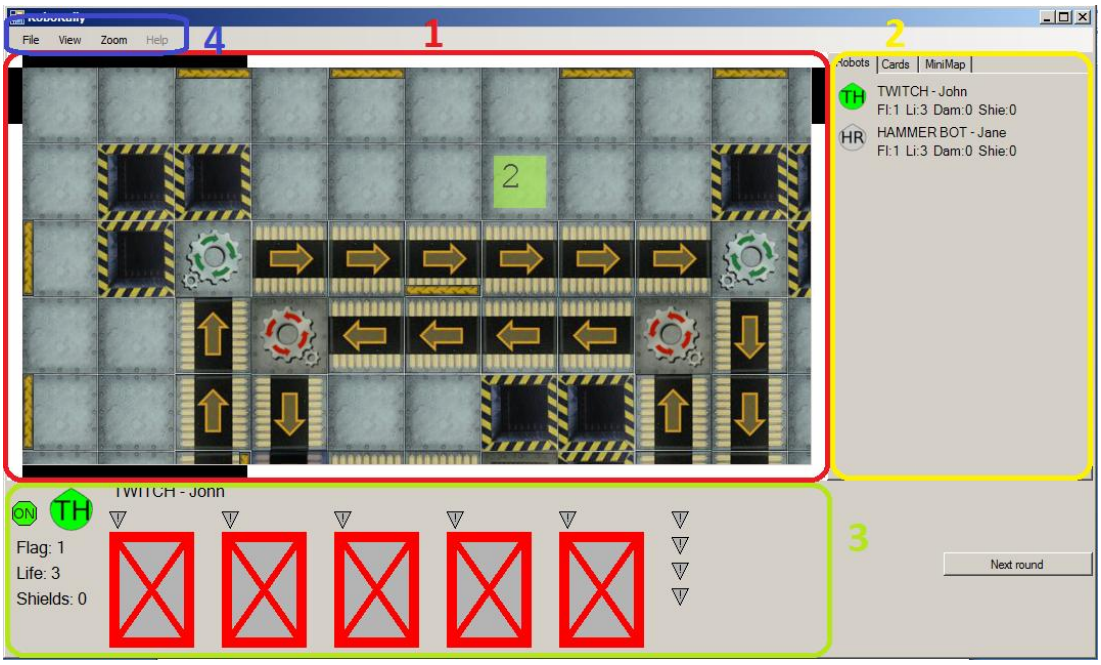
   c) *Computer* – Robot se bude účastnit hry a bude ovládán počítačem.

3) *Start position* – Startovní pozice, na které se robot objeví na začátku hry. Každý robot musí mít jinou startovní pozici. (V mapě jsou startovní pozice vyznačeny černými číslu v kruhu.)

Hry se musí účastnit alespoň dva roboti. Po nastavení robotů pokračujte stiskem *OK* okna *Robots*. Pokud roboty nenastavíte správně, (např.: dva hrající roboti budou mít stejné jméno nebo stejnou startovní pozici) budete na tuto skutečnost upozorněni dialogovým oknem a nastavení bude nutné před pokračováním změnit tak, aby odpovídalo uvedeným požadavkům. Pokud jste roboty nastavili správně, bude po stisku tlačítka *OK* vytvořena nová hra. V případě, že se hry účastní roboti ovládaní počítačem, může akce vytvoření hry jistou chvíli trvat, v závislosti na počtu takovýchto robotů a na konfiguraci počítače, na kterém je program spuštěn. Po vytvoření hry je aktivováno hlavní okno aplikace s definovanou hrací plochou.

A.4 Hra

A.4.1 Okno hry



Obrázek 14 Okno hry

Okno hry se skládá z následujících částí:

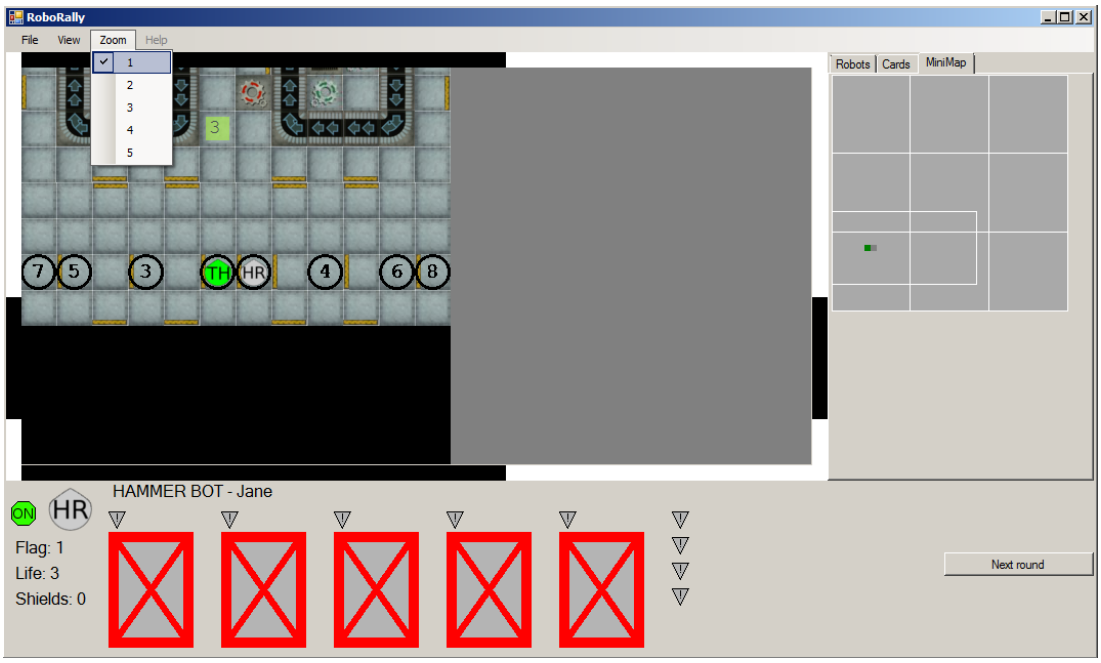
1) Mapa
2) Panel se záložkami
3) Informace o zvoleném robotovi
4) Menu

Hlavní část plochy okna zabírá zobrazení mapy. Mapa se skládá z jednotlivých políček, velikost jednoho políčka lze nastavit v menu *Zoom*. Po zvětšení políček je možné si mapu lépe prohlédnout, naopak po jejich zmenšení je zobrazena větší část mapy najednou. Jelikož se mapa do okna většinou nevejde celá, je možné s obrázkem mapy pohybovat. Pohyb mapou lze provést buď posuvníky na okraji, které zároveň ukazují, jaká část mapy je viditelná, nebo stisknutím a držením levého tlačítka myši nad obrázkem mapy a tažením.

Napravo od mapy je panel se záložkami *Robots*, *Cards* a *MiniMap*. Na záložce *Robots* je zobrazen seznam robotů. U každého z nich je jeho obraz, jeho jméno a jméno hráče, který ho ovládá a základní informace (*Fl* – vlajka na kterou robot jde, *Li* – počet životů, *Dam* – poškození, *Shie* – štíty). Kliknutím na některého robota v seznamu, se daný robot vybere a jeho informace se zobrazí v panelu pod mapou.

Záložka *Cards* se využije až při programování robota. Na záložce *MiniMap* je zmenšenina mapy, umožňující rychlé zobrazení libovolné části mapy. Ve zmenšenině nejsou vidět jednotlivá políčka mapy, ale pouze jednotlivé desky, ze kterých se mapa skládá. Desky jsou zde zobrazeny jako devět pravidelných bílých čtverců (bez výplně) na šedém pozadí. Dále je obdobně bílým obdélníkem zvýrazněno, která část mapy je právě zobrazena. Malými barevnými čtverci jsou zde vyznačeny aktuální pozice robotů. Kliknutím, případně stiskem a držením levého tlačítka myši se zobrazí část mapy, nad kterou je kurzor.

V hlavním okně pod mapou je panel zobrazující podrobné informace vybraného robota. V levém horním rohu je obrázek, který je buď zelený, nebo červený, podle toho zda bude daný robot příští kolo zapnutý nebo vypnutý. Vpravo od něj je obrázek robota, jméno robota a jméno hráče. Pod obrázkem robota je postupně uvedeno na jakou vlajku robot právě jde, kolik má životů a kolik má štítů. Dále panel obsahuje pět větších obdélníků, tam jsou zobrazeny karty robota, pokud má nějaké zamčené. V okolí míst pro karty je devět trojúhelníků. Pokud je robot poškozen, tak za každé jeho poškození je jeden trojúhelník vybarven žlutě.



Obrázek 15 Ukázka zmenšené mapy a MiniMapy

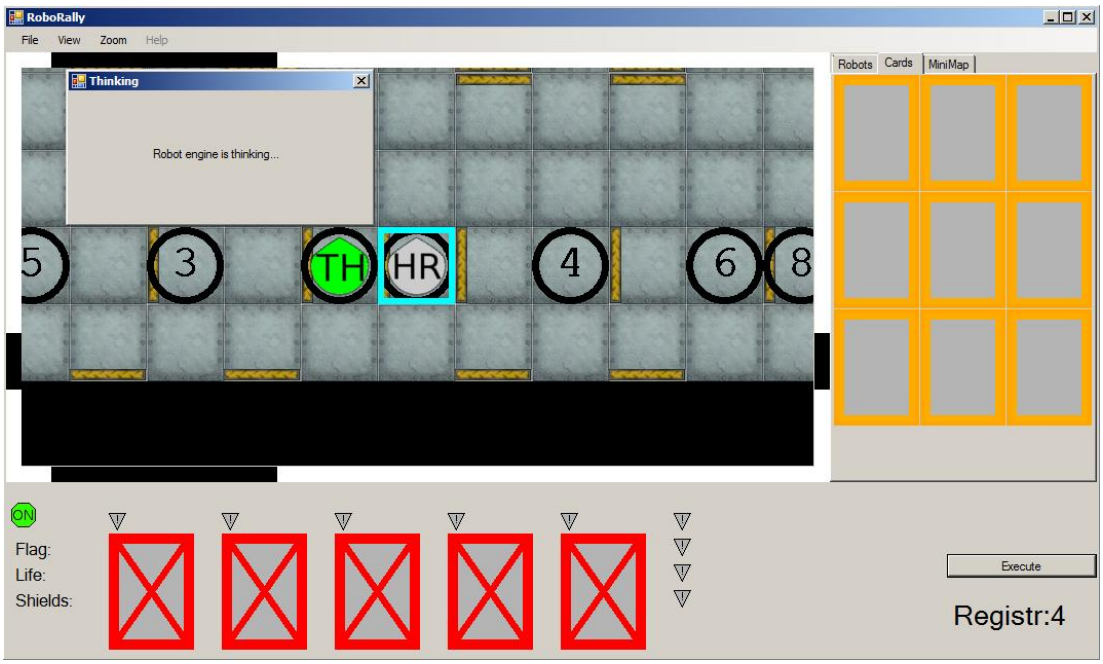
A.4.2 Hra

Hra probíhá v jednotlivých kolech. V každém kole nejprve všichni hráči naprogramují svého robota (tj. stanoví sekvenci pěti karet) a poté se postupně všech pět karet provede.

Pro zahájení programování robotů stiskněte tlačítko *Next round* v pravém dolním rohu hlavního okna. Tím se zahájí další kolo hry a přejde se k programování prvního robota.

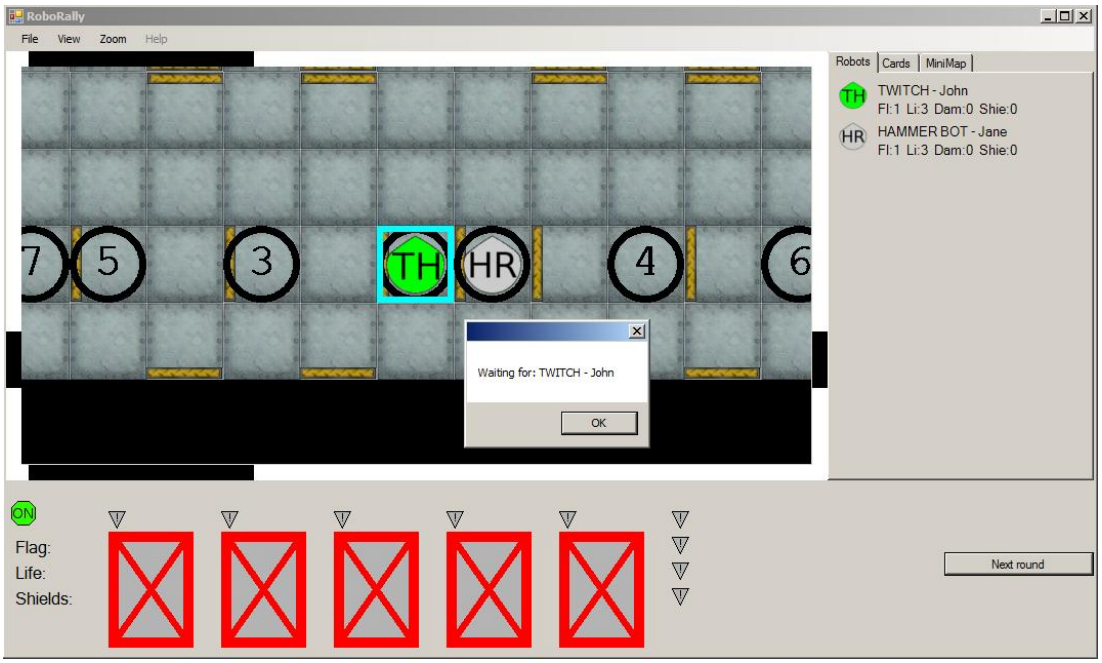
A.4.2.1 Programování robotů

Programování každého z robotů probíhá následovně. Pokud je robot ovládán počítačem, zobrazí se okno s informací, že robot přemýšlí (pokud je počítač rychlý, okno jen problikne). Po skrytí informačního okna, je pro zahájení programování dalšího robota nutné přejít stiskem tlačítka *Next player*, v pravém dolním rohu.



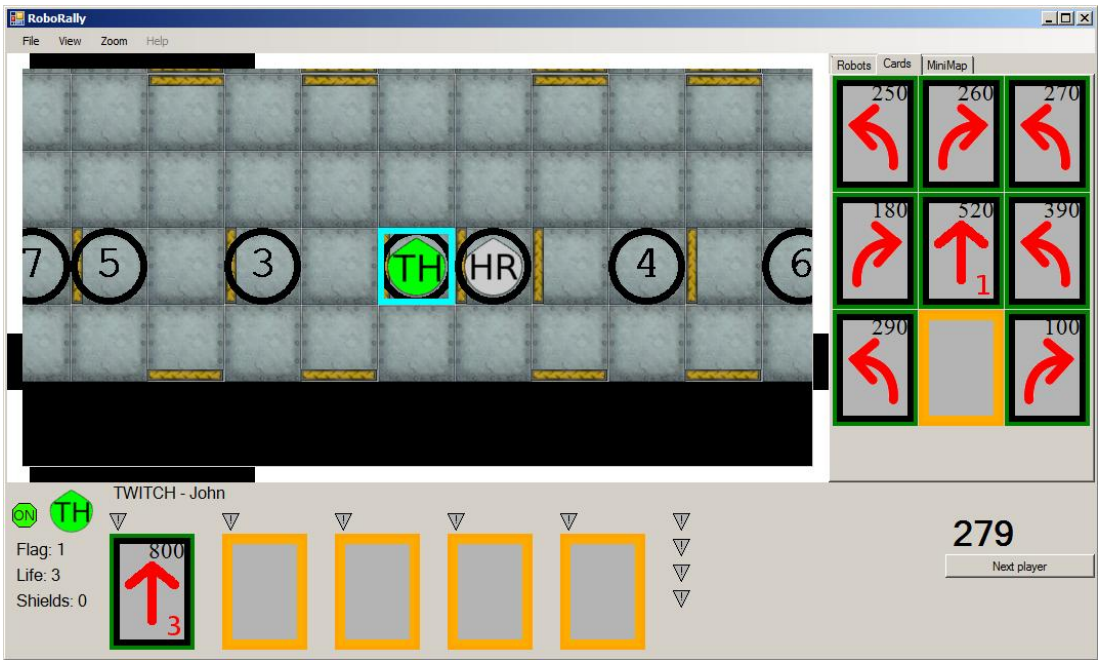
Obrázek 16 Programování robota ovládaného počítačem

Pokud je aktuálně programovaný robot ovládán člověkem, zobrazí se nejprve dialogové okno, které čeká na stisk tlačítka *OK*.



Obrázek 17 Dialogové okno s čekáním na hráče

Po stisku OK v dialogovém okně může hráč začít programovat svého robota.



Obrázek 18 Okno s programováním robota

Programování probíhá v hlavním okně. Při zahájení programování konkrétního robota, je mapa zaměřena na právě na tohoto robota, který je v mapě navíc zvýrazněn světle modrým rámečkem. Zároveň se v pravém dolním rohu odpočítává čas, který hráči na dokončení programování zbývá. Na panelu vpravo je zobrazen obsah záložky *Cards*. Ta obsahuje devět míst pro karty daného hráče. V prvním kole hry obdrží každý hráč devět karet, v dalších kolech potom o tolik méně karet kolik má poškození. Panel pod mapou je zaměřen na právě programovaného robota. V něm jsou prázdné, oranžově ohraničené, rámečky do kterých hráč musí umístit některé z karet z panelu vpravo nahoře.

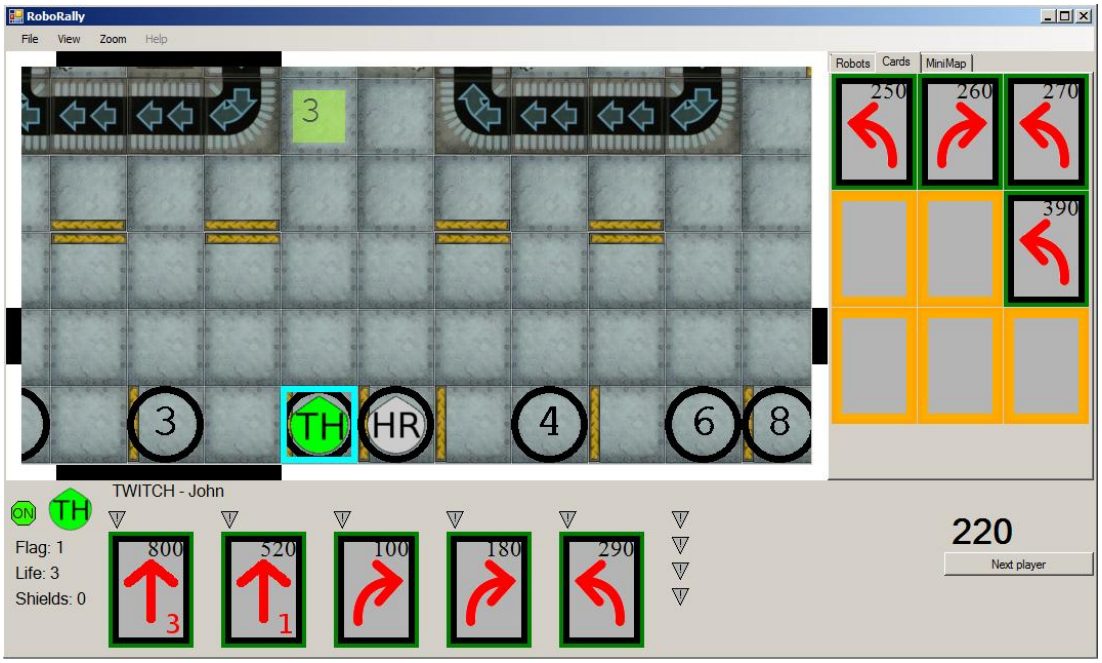
Umístění karet do prázdných rámečků se provede tak, že hráč nad nějakou kartou stiskne a drží levé tlačítko myši, a následně přetáhne myš nad prázdný rámeček, kde tlačítko myši pustí. Takto lze karty přemisťovat i mezi libovolnými pozicemi pro karty, kromě těch, na kterých jsou karty zamčeny. Zamčené karty se poznají podle toho, že mají okolo sebe červené orámování, na rozdíl od normálního zeleného. Kartu lze přesunout i do rámečku, kde již karta je, v tom případě se karty mezi danými dvěma rámečky/pozicemi prohodí.

Hráč musí vyplnit všechny prázdné rámečky svého robota na spodním panelu, to jsou jeho registry. Navíc ještě může chtít svého robota vypnout pro příští kolo. To provede stiskem zeleného šestiúhelníkového tlačítka s nápisem *ON* v levém horním rohu spodního panelu. Tlačítko se tím změní na červené s nápisem *OFF*. Vypnutí

robota pro příští kolo znamená, že hráč svého robota příští kolo nebude programovat, a ten tedy ani nebude provádět žádné své pohyby. Za to se robotovi na začátku kola, kdy je vypnutý vymažou všechna poškození.

Během programování robota může hráč prohlížet mapu a svého robota nebo, po přepnutí panelu vpravo nahoře na záložku *Robots*, i jiné roboty. K programování svého robota se vrátí jeho vybráním na záložce *Robots*. Pokud hráči vyprší časový limit, který má k dispozici na programování svého robota, nebo po stisku tlačítka *Next robot* v pravém dolním rohu, přejde se k programování dalšího robota. Pokud hráč svému robotovi nestihl do vymezeného časového limitu vyplnit všechny jeho registry, jsou mu nevyplněné registry doplněny náhodně ze zbývajících karet, které nepoužil.

Pokud je právě programován poslední robot, tlačítko v pravém dolním rohu má popisek *Execute*. Po stisku tohoto tlačítka se zahájí jednotlivé fáze provádění registrů.



Obrázek 19 Okno se všemi registry robota vyplněnými

A.4.2.2 Provádění registrů robotů

Provádění naprogramovaných registrů probíhá v pěti fázích, pro každý registr jedna. Každá fáze postupuje v následujících krocích:

1) Postupně každý robot provede pohyb zadaný kartou, kterou má v registru pro danou fázi. Roboti pohyby provádějí v pořadí podle priority karet pro aktuální

fázi. Začíná robot, jehož karta má nejvyšší prioritu. (Každá karta má svou prioritu napsánu v pravém horním rohu.)

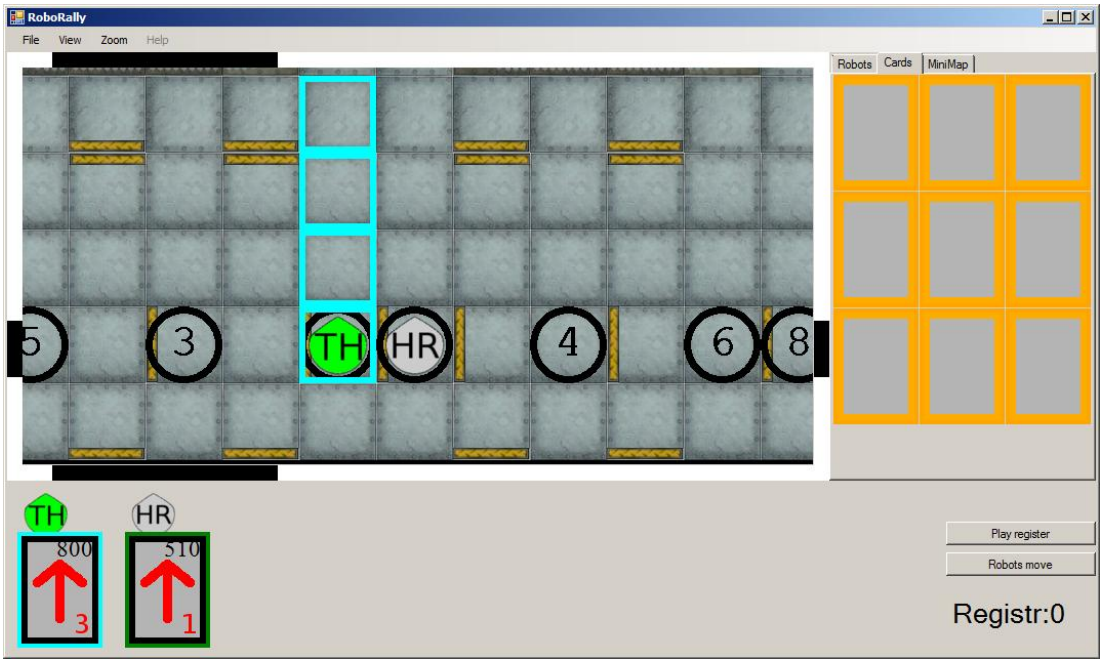
2) Pohnou se expresní dopravníky. (Funguje stejně jako v bodě tři, ale pouze na expresní dopravníky.)
3) Pohnou se expresní i obyčejné dopravníky. Každý robot, který stojí na nějakém dopravníku, je posunut o jedno políčko v jeho směru. Ale pouze pokud tomu něco nebrání. Pokud je robot posunut na navazující dopravník, který zatáčí, je robot tímto dopravníkem otočen, podle toho jak dopravník zatáčí.
4) Strkače zatlačí. Každý robot stojící před strkačem, který je v této fázi aktivní, je jím posunut o jedno políčko. Pokud posunu brání jiní roboti, jsou posunuti také.
5) Převodovky se otočí. Každý robot stojící na převodovce je jí otočen ve směru jejích šipek.
6) Střelba laserů. Ze všech laserů v mapě a z laserů robotů, je vystřeleno. Pokud v cestě vystřeleného laseru stojí robot (laser neprojde zdí), je laserem zasažen. Každý robot získá za každý zásah jedno poškození. Pokud již robot získal devět poškození a je zasažen laserem, zemře.
7) Dotyk vlajek a kontrolních bodů. Pokud nějaký robot v tomto bodě stojí na vlajce, na kterou má jít, má splněno a může pokračovat na další vlajku v pořadí. Pokud již robot prošel všechny vlajky ve správném pořadí, dokončil celou hru. Každý robot, který v tomto bodě stojí na nějaké vlajce nebo opravně si na ni umístí svou uloženou pozici. (To je pozice, na kterou se vrátí, pokud zemře, ale zbude mu ještě alespoň jeden život.)

Pro následující odstavce bude „akce“ znamenat buďto provedení karty robotem v kroku jedna nebo kterýkoliv další celý krok fáze.

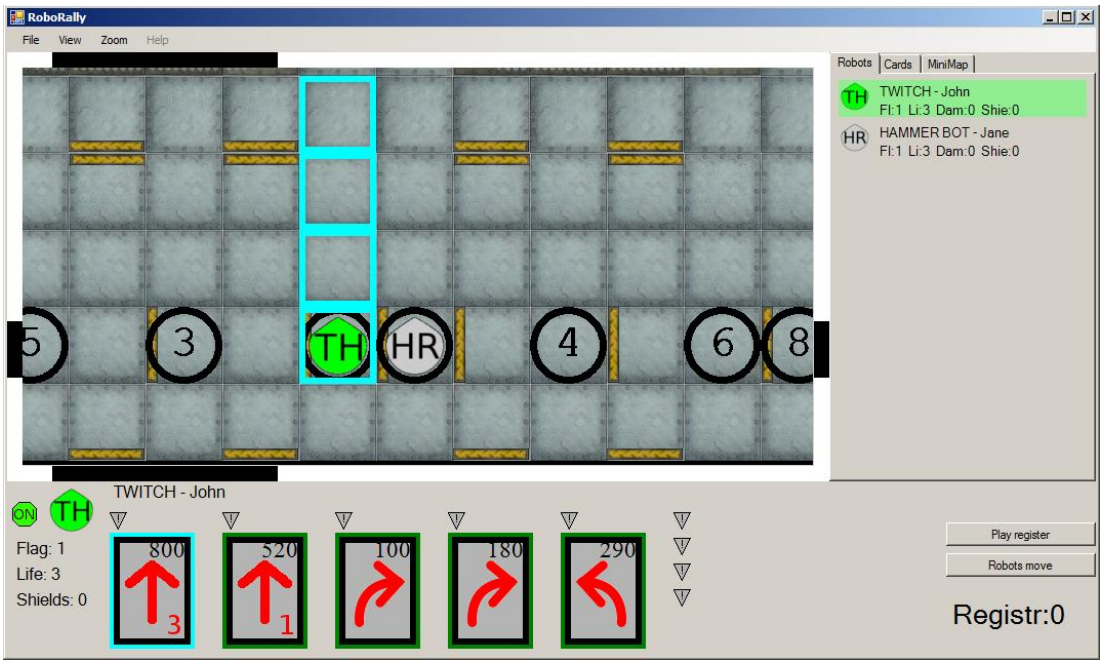
Při provádění registrů robotů jsou vpravo dole zobrazena dvě tlačítka, pod kterými je nápis informující o tom, jaký registr je aktuálně prováděn. Horní z tlačítek má vždy nápis *Play register*. Stisknutím tohoto tlačítka se dokončí provádění aktuálního registru. To znamená, že se provedou jednotlivé body zbývající k dokončení aktuálního registru. Hráčům se ukáže výsledek každé akce, která změní stav hry (např. pohyb některého robota). Zvýraznění je provedeno tak, že se nejprve ohraničí rámečkem všichni roboti, kterých se změna přímo týká a pokud se roboti budou pohybovat z daného políčka, jsou označena i políčka přes která se budou

pohybovat. Pokud je to možné, mapa je posunuta, případně zmenšena tak, aby všechna zvýrazněná políčka byla vidět. Po zvýraznění a posunutí mapy je krátká časová prodleva, aby hráči mohli vidět, kteří roboti a políčka budou akcí ovlivněna. Poté se provede příslušná akce a znovu nastane krátká časová prodleva tak, aby byl vidět výsledek dané akce. Takto jsou postupně vizualizovány všechny akce, které mění stav hry.

Popisek na spodním tlačítku v pravém dolním rohu okna se postupně mění a označuje, jaká akce se po jeho stisku provede. Opakovaným stiskem spodního tlačítka se provedou všechny kroky aktuální fáze a to i v případě, že se v nich nemění stav hry. Pokud stisk tlačítka představuje změnu, jsou už před jeho stiskem označena políčka, na kterých se bude něco dít.

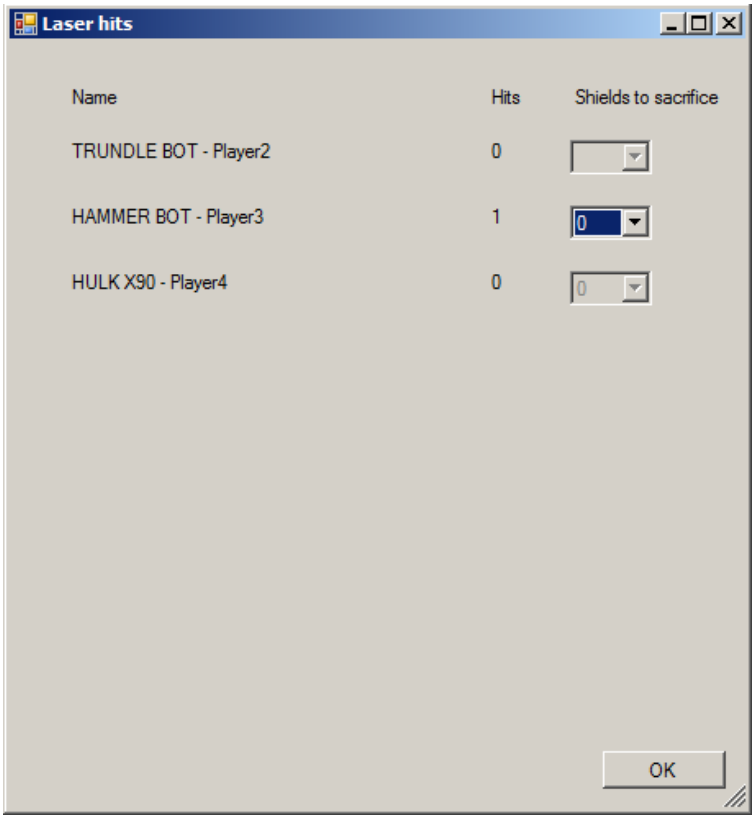


Obrázek 20 Okno hry před provedením pohybu robota



Obrázek 21 Zobrazení stavu robota, během provádění registrů

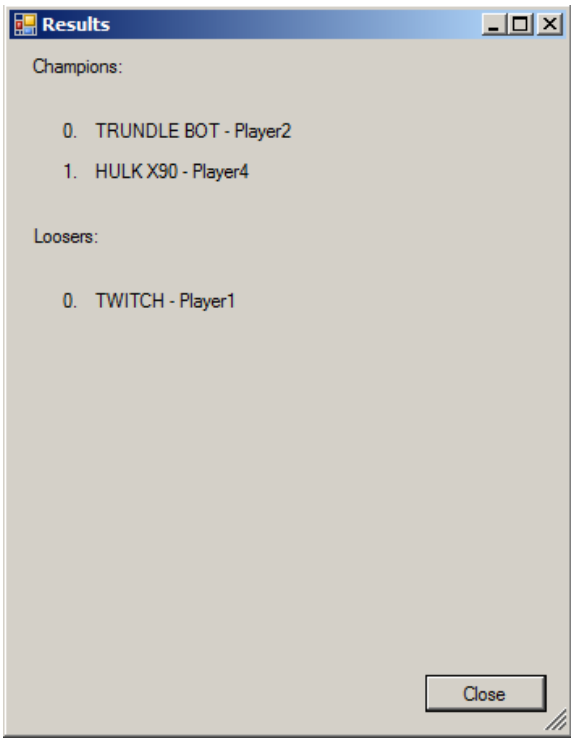
Pokud je při střelbě laserů zasažen některý z robotů, kteří mají nenulový počet štítů, zobrazí se dialogové okno. V něm jsou v tabulce zobrazeni všichni roboti. Ti z nich, kteří byli zasaženi laserem a mají nějaké štíty, si zde mohou zvolit, kolik štítů chtějí obětovat. Pokud robot obětuje štít, neuplatní se na něm zásah laserem.



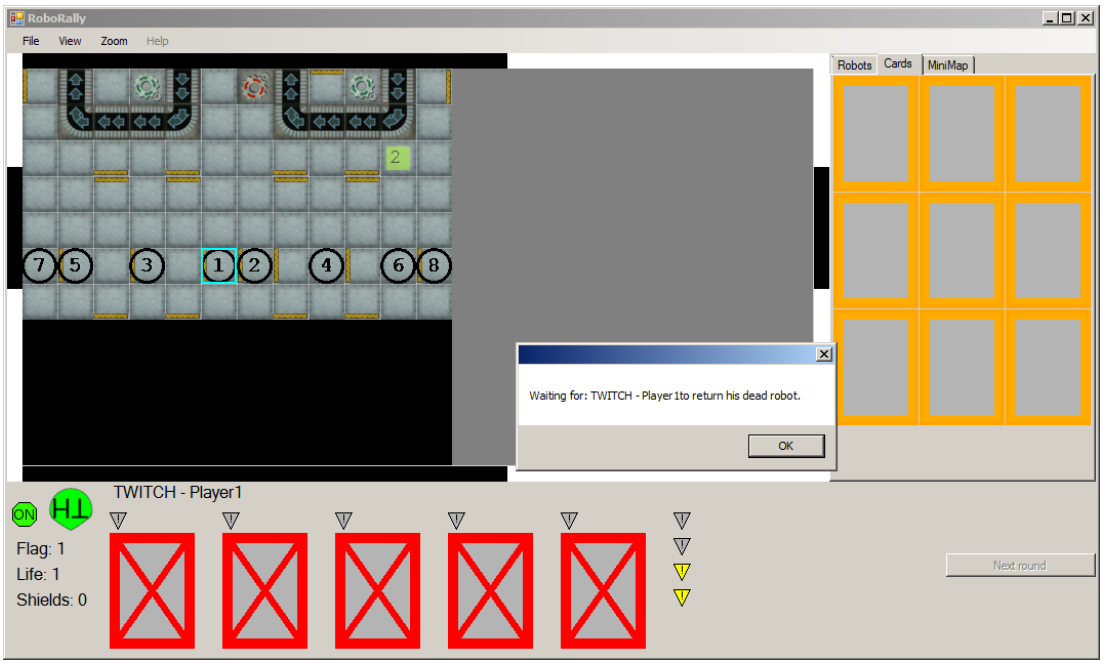
Obrázek 22 Tabulka se zásahy lasery

Po provedení všech pěti registrů se zobrazí tabulka robotů, kteří již prošli všechny vlajky a těch, kterým došly všechny životy. Poté proběhne ještě tzv. úklid. Každému robotovi, který při něm stojí na vlajce, nebo opravně je odebráno jedno poškození. Každý robot, který stojí na opravně s klíčem a kladivem dostane navíc jeden štít. Po tomto jsou ještě do hry vráceni roboti, kteří toto kolo zemřeli, ale ještě jim zbývá alespoň jeden život. Roboti řízení počítačem jsou do hry vráceni bez interakce s hráči. Pro každého robota, který se vrací do hry a je ovládán hráčem se zobrazí dialogové okno, které o této skutečnosti informuje. Po jeho zavření je mapa posunuta tak, aby byla vidět políčka, na která hráč může svého robota umístit. Robota lze umístit na políčko, kde má uloženou pozici, nebo pokud na tomto políčku již je jiný robot, na libovolné sousední políčko, které není bezedná jáma. Před tím než hráč svého robota umístí, ještě mu může nastavit, že bude vypnutý, toto nastavení se uplatní hned následující kolo. Hráč svého robota umístí dvojklikem na některé zvýrazněné políčko. Po umístění robota na políčko, je ještě zobrazeno dialogové okno, ve kterém hráč určí, kterým směrem bude jeho robot natočen.

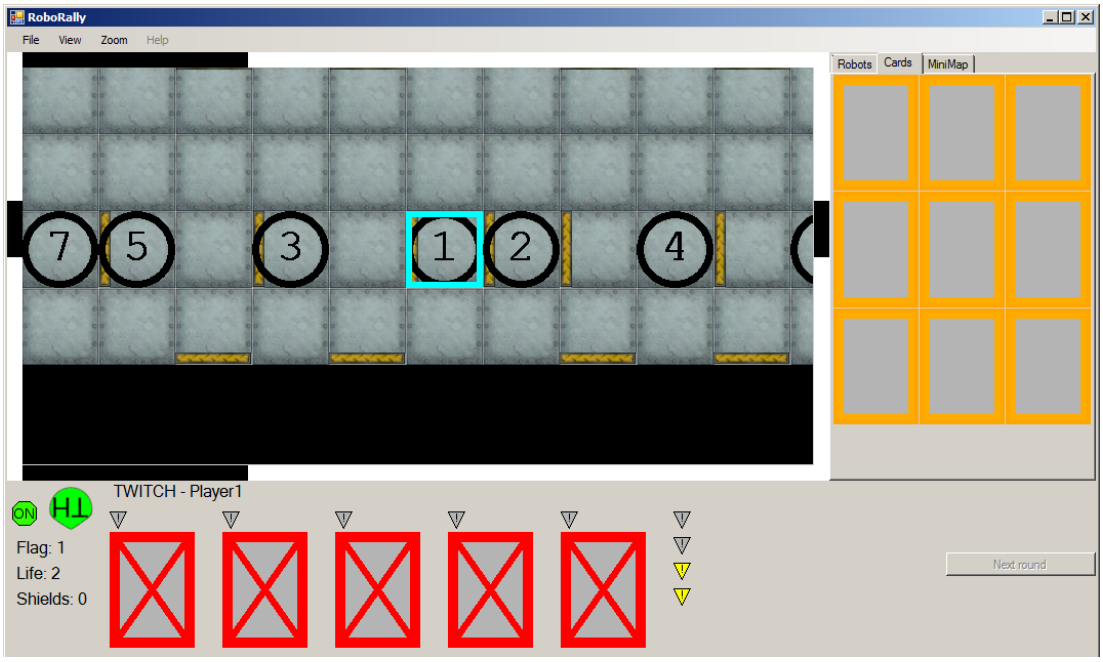
Když je úklid dokončen, přejde hlavní okno hry do stavu jako před programováním robotů. V tomto stavu si lze prohlížet mapu a celou hru. Další kolo programování se spustí opět stiskem tlačítka *Next round*.



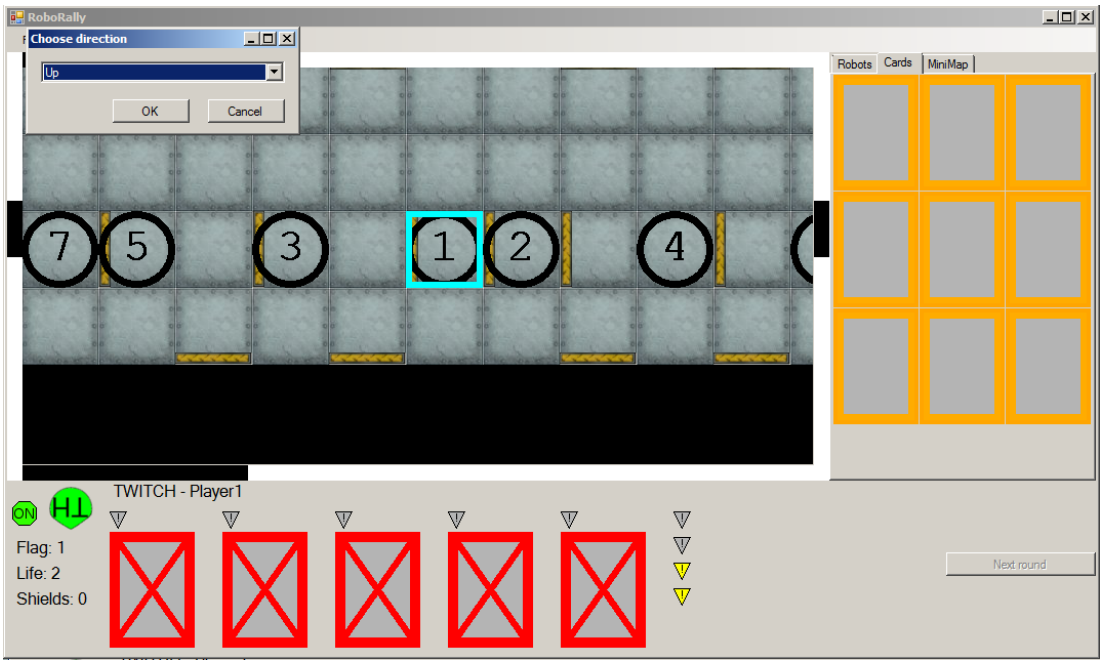
Obrázek 23 Tabulka robotů co skončili



Obrázek 24 Čekání na hráče, aby vrátil do hry svého robota



Obrázek 25 Vracení robota do hry



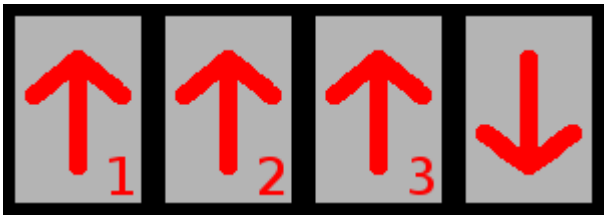
Obrázek 26 Výběr směru, jakým bude robot natočen

A.5 Podrobnější popis součástí hry

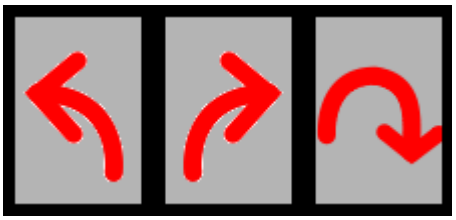
A.5.1 Druhy karet

Ve hře je sedm druhů karet. Čtyři druhy karet robota posunou a to buď o jedno, dvě nebo tři políčka vpřed, nebo o jedno políčko vzad. Další tři druhy karter robota nepředstavují žádný posun, ale mění směr, kterým je robot natočen. Jedná se o otočení o 90° vlevo (tj. proti směru hodinových ručiček), vpravo (tj. po směru hodinových ručiček) a otočení vzad. Každá karta má ve svém pravém horním rohu vyznačenu prioritu. Ta při provádění daného registru určuje, v jakém pořadí roboti své pohyby provedou. Začíná se od karty s nejvyšší prioritou.

Když se robot posunuje díky kartě pohybu, dělá to vždy postupně, po jednotlivých políčcích. Pokud mu v cestě brání jiný robot, je posunut také. Jestliže ale robot narazí na zeď, nebo na ni narazí jiný robot, který je také zároveň posouván, pohyb robota/robotů se zastaví. Pokud je v cestě posunu robota bezedná jáma, robot do ní spadne.



Obrázek 27 Karty pohybu vpřed o jedno, dvě a tři políčka, karta pohybu vzad

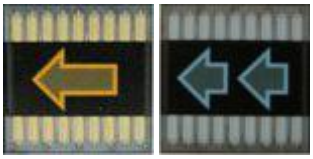


Obrázek 28 Karty otočení vlevo, vpravo a vzad

A.5.2 Druhy políček

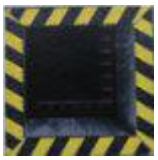
Obrázek 29 Obyčejné políčko

Obyčejné políčko robota nijak neovlivňuje.



Obrázek 30 Dopravník, expresní dopravník

Pokud robot po provedení pohybu robotů stojí na expresním dopravníku, je posunut ve směru jeho šipek, pokud tam není jiná překážka. Pokud robot po provedení pohybu expresních dopravníků stojí na dopravníku, nebo expresním dopravníku, je také posunut ve směru jeho šipek. Když nějaký dopravník posune robota na jiný dopravník, který zatáčí, je robot ihned pootočen podle toho kam daný dopravník zatáčí.



Obrázek 31 Bezedná jáma

Pokud se robot kdykoliv ocitne na políčku bezedné jámy, okamžitě je mu odebrán jeden život a je odstraněn z hracího plánu. Do hry se vrátí až pro příští kolo, pokud mu ještě zbyly nějaké životy. Stejně jako bezedná jáma působí mimo hrací plán, nebo posun na černé políčko.

Obrázek 32 Převodovka otáčející se ve směru hodinových ručiček

Jestliže robot stojí na políčku převodovky při kroku otáčení převodovek, je otočen o 90° ve směru jejího otáčení.

Obrázek 33 Laser, stěna, opravna s kladivem i klíčem, opravna, vlajka, strkač

Políčko laseru způsobí střelbu laseru ve vyznačeném směru v příslušném kroku fáze. Robot stojící nejblíže laseru (v daném směru), když laser vystřelí, je zasažen. Lasery mohou být jednoduché, dvojité i trojité, robot tak může obdržet jeden, dva, případně tři zásahy. Každý robot má navíc svůj laser, kterým střílí ze svého políčka ve směru kam je natočen.

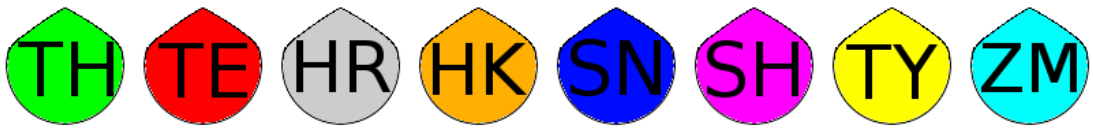
Políčko obsahující stěnu slouží k omezení pohybu robotů a dosahu laserů v herním plánu. Pokud se ve směru pohybu robota nachází stěna, pohyb robota se zastaví. Rovněž paprsek laseru stěnou neprojde.

Políčka opravny jsou dvojího druhu. Pokud robot stojí na opravně po konci kola, je mu za to odebráno jedno poškození. Pokud navíc stojí na vylepšující opravně (opravna s kladivem a klíčem), obdrží navíc jeden štít.

Políčko vlajky obsahuje zelený čtverec s číslem vlajky.

Jestliže se robot nachází na políčku se strkačem, v kroku aktivity strkačů tento strkač odtlačí robota na sousední políčko (dle směru strkače).

A.5.3 Roboti



Obrázek 34 Obrázky jednotlivých robotů

Všichni roboti mají stejný kulatý tvar s jednou špičkou, která ukazuje směr natočení robota. Všichni roboti jsou rovnocenní, liší se jen svými obrázky. Každý

robot obsazuje na hracím plánu jedno políčko, pokud se robot pohybuje na políčko, kde je již jiný robot, buďto se o něj zasekne, nebo jej vytlačí. Každý robot má svůj laser, ten střílí z políčka kde je robot umístěn směrem, kam je robot natočen. Roboti mají následující vlastnosti:

1) Vlajka – následující vlajka, kterou se robot musí snažit dosáhnout.
2) Životy – pokaždé když robot zemře, je mu odebrán jeden život, pokud už žádný život nezbude, robot končí. Na začátku hry může mít robot podle nastavení tři až pět životů.
3) Štíty – pokud má robot nějaké štíty a je zasažen laserem, může je obětovat a díky tomu se ubránit poškození. Při zahájení hry je počet štítů každého robota nulový.
4) Poškození – pokud je robot zasažen laserem, je mu přidáno poškození. Za každé páté a další poškození je robotovi zamčen jeden registr, pak není možné s kartou v daném registru hýbat. Pokud robot získá devět poškození a je znovu zasažen laserem, zemře.
5) Uložená pozice – každý robot má někde uloženou pozici. Při zahájení hry je uloženu pozicí startovní pozice daného. Pokud robot po provedení některého registru skončí na opravně, nebo vlajce, umístí se jeho uložená pozice na dané políčko. Pokud robot zemře a vrací se do hry, vrací se vždy na svou uloženou pozici.

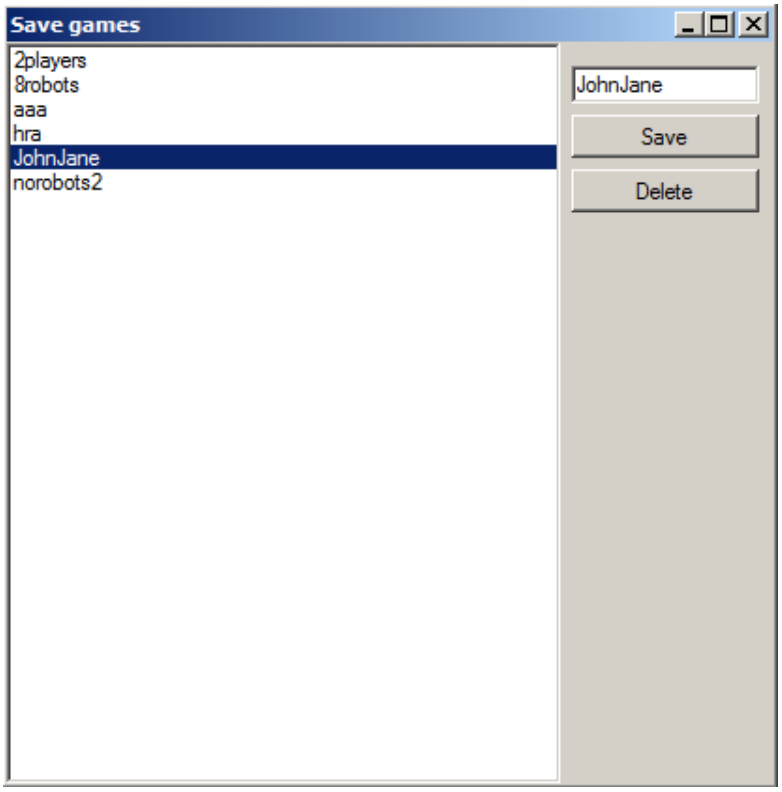
A.6 Uložení a načtení hry

A.6.1 Uložení hry

Aktuálně rozehranou hru lze uložit. To lze provést v případě, že je nějaká hra rozehraná a právě neprobíhá programování robotů nebo provádění registrů. Uložení hry provedete volbou položky *Save game* v menu *File* hlavního okna. V levé části zobrazeného dialogu *Save games* je seznam již uložených her. Výběrem položky v tomto seznamu se box vpravo vyplní jménem vybrané položky, stejně tak napsáním jména uložené hry do boxu se v seznamu daná položka vybere.

Pro uložení hry je nutné mít v boxu jméno, pod jakým chceme hru uložit a stisknout tlačítko *Save*. Pokud uložená hra s daným jménem neexistuje, hra se uloží, pokud již existuje, budete nejprve dotázáni, jestli si přejete již existující

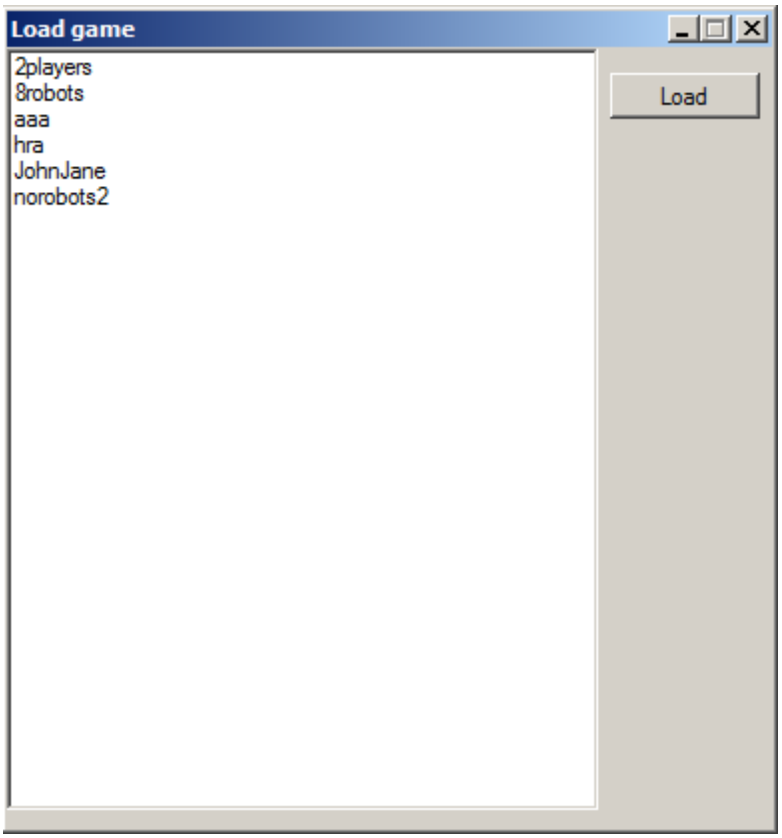
uloženou hry přepsat. Dialog umožňuje i smazání již existující uložené hry, to provedete vybráním hry v seznamu a stiskem *Delete*.



Obrázek 35 Dialog uložení hry

A.6.2 Načtení hry

Načtení dříve uložené hry provedete v dialogovém okně, které se zobrazí po výběru položky *Load game* v menu *File* hlavního okna. Zde v seznamu uložených her vyberte, kterou chcete načíst a pro její načtení stiskněte *Load*. Pokud máte právě nějakou hru rozehranou, budete ještě dotázáni, zda jste si jisti jejím ukončením.



Obrázek 36 Dialog načtení hry

A.7 Pokročilé sestavení herního plánu

A.7.1 Vytvoření herního plánu

Namísto načtení již přednastavené mapy (herního plánu), jak již bylo ukázáno dříve, je možné vytvořit si vlastní mapu. K tomu slouží okno *Create map*, které zobrazíte výběrem položky *New game* v menu *File* hlavního okna. V okně *Create map* je možné sestavit si mapu z až devíti desek. Desky, ze kterých je možné mapu sestavit, jsou v pravém sloupci. Na levé straně je matice 3x3 šedých čtverců. Každý z těchto čtverců je místo, kam lze vložit desku. Desku z pravého sloupce umístíme do mapy tak, že nad jejím obrázkem stiskneme a držíme levé tlačítko myši, tím se vybraná deska zvýrazní modrým rámečkem, pak myš přesuneme nad jeden z devíti čtverců a pustíme, tím se do něj deska umístí.

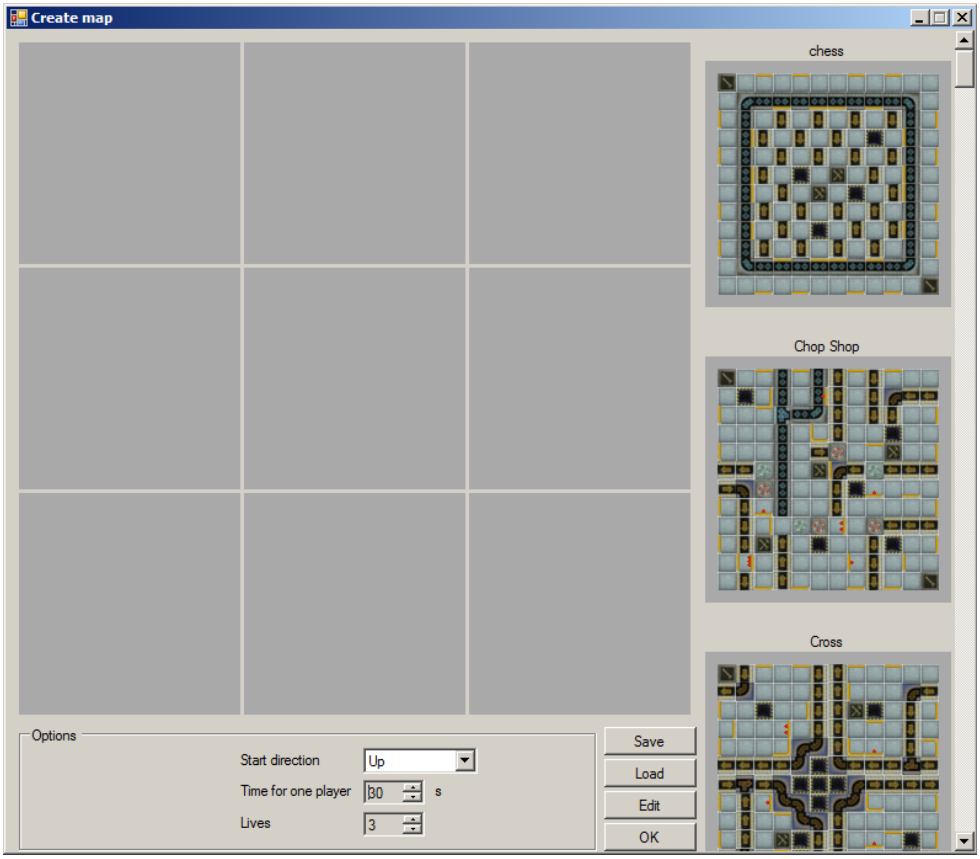
Každou vloženou desku lze samozřejmě zase odebrat, nebo ji pootočit. Stiskem pravého tlačítka myši nad již vloženou deskou se zobrazí vyskakovací menu, které toto umožňuje.

Aby byla mapa kompletní, je nutné do ní ještě vložit alespoň jednu vlajku a všech osm startovních pozic. Stiskem tlačítka *Edit* se zobrazí okno *Map editing*,

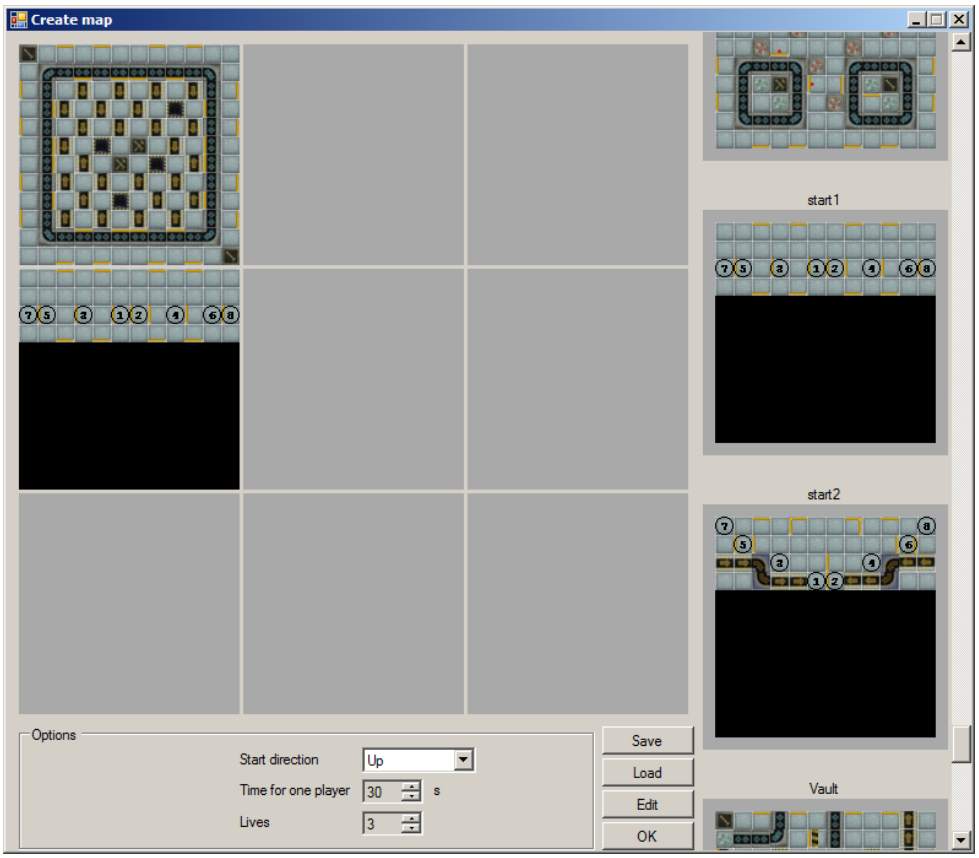
s načtenou mapou tak, jak ji právě máme v okně *Create map*, které vkládání vlajek a startovních pozic umožňuje. V jeho levé části je zobrazena mapa, s tou lze manipulovat metodou drag&drop, stejně jako s mapou zobrazenou při hraní hry. Vpravo od mapy jsou v jednom sloupci vlajky a ve druhém startovní pozice. Ty se do mapy umisťují stejným způsobem. Nad obrázkem vlajky nebo startovní pozice stiskněte a držte levé tlačítko myši, pak myš přetáhněte nad mapu. Políčko, nad kterým je kurzor myši, se vždy zvýrazní modrým rámečkem, po uvolnění tlačítka myši se vlajka nebo startovní pozice umístí na zvýrazněné políčko.

Pokud umístíme již položenou vlajku nebo startovní pozici na jiné políčko, tato se přesune z původního místa. Pro odstranění vlaky nebo startovní pozice z mapy úplně na její políčko klikněte pravým tlačítkem, je zobrazeno vyskakovací menu s jedinou položkou nabízející odstranění. Vlajku ani startovní pozici nelze vložit na políčko, kde již je jiná vlajka nebo startovní pozice, nebo na políčko s bezednou jámou. Takové políčko se tedy ani nezvýrazní, když na něj zkusíme vlajku či startovní pozici položit.

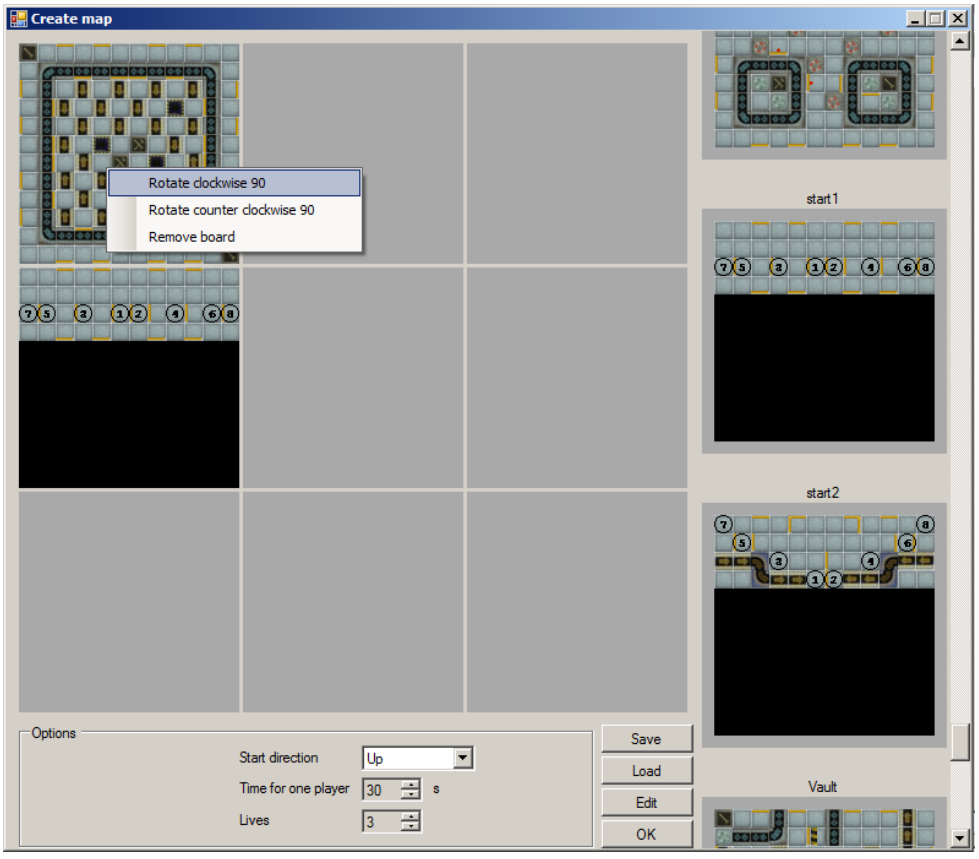
Pro vložení startovních pozic je nejjednodušší vložit do mapy desku, která již startovní pozice obsahuje.



Obrázek 37 Okno sestavování herního plánu



Obrázek 38 Sestavený herní plán



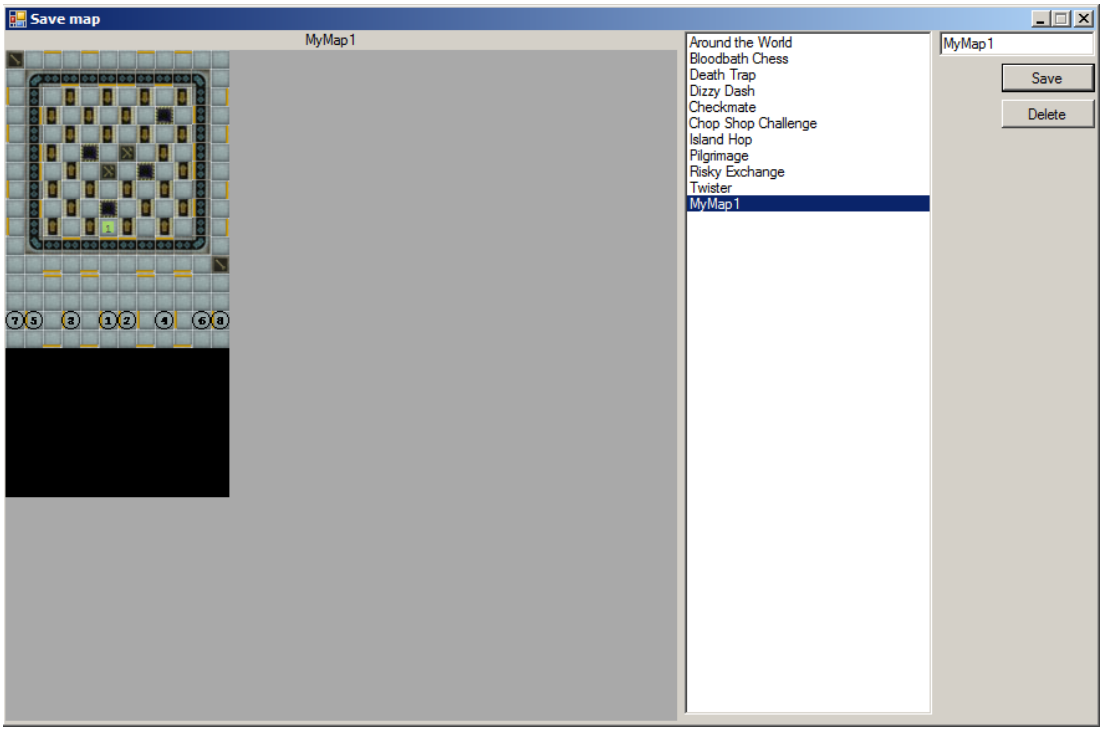
Obrázek 39 Vyskakovací menu v Create map

Obrázek 40 Úprava vlajek a startovních pozic

Obrázek 41 Odstranění vlajky

A.7.2 Uložení herního plánu

Pokud máte v okně *Create map* vytvořený zajímavý herní plán, pravděpodobně ho budete chtít uložit a někdy v budoucnu znovu použít. Pro uložení plánu stiskněte tlačítko *Save*. Tím se zobrazí okno *Save map*, to funguje stejně jako okno *Save games*, které slouží pro ukládání rozehraných her. Pro uložení mapy buďto vyberte ze seznamu název existující mapy, kterou chcete přepsat, nebo zadejte nové jméno do boxu vpravo. Vlastní uložení se provede po stisku tlačítka *Save*, tím se mapa uloží a zároveň se přidá do seznamu uložených map. Pokud název ukládané mapy vyberete ze seznamu (vyberete název existující mapy), budete ještě dotázání, zda opravdu chcete původní mapu přepsat. Jednotlivé uložené mapy si zde můžete i prohlížet, výběrem mapy v seznamu se v levé části zobrazí vizuální podoba mapy. Když budete chtít některou z uložených map odstranit, stačí ji vybrat v seznamu a stisknout tlačítko *Delete*, samozřejmě budete ještě dotázání, zda jste si smazáním mapy jisti. Načtení již uložené mapy provedete stiskem tlačítka *Load* v okně *Create map*. Aktivity spojené s načtením dříve uložené mapy jsou popsány v sekci *Jak začít hrát*.



Obrázek 42 Okno ukládání herního plánu

A.8 Záznam

V menu *View* lze zobrazit okno se záznamem hry. Toto okno obsahuje pouze textové pole, kam je vypisováno některé dění ve hře. Například, jaké karty hráči dostali a jaké si vybrali. (Do záznamu nejsou ukládány všechny změny ve hře.)

Příloha B: Programátorská dokumentace

Tento dokument obsahuje programátorskou dokumentaci k aplikacím vytvořeným v rámci bakalářské práce RoboRally. Celý systém byl vytvořen ve vývojovém prostředí Visual Studio 2010 v jazyce C# pro operační systém Windows. Je postaven na platformě .NET 4.0 a sestává z následujících programů:

4) RoboRally – dále jen RoboRally GUI. Okenní aplikace umožňující hraní hry RoboRally na PC ve více hráčích, nebo proti počítači dle zadání práce.
5) Evolution2 – konzolová aplikace, která byla vytvořena za účelem hledání vhodných nastavení parametrů umělých inteligencí (enginů) robotů.
6) RoboRally_Console – konzolová aplikace pro hraní hry RoboRally mezi roboty ovládanými počítačem.

B.1 Herní jádro

Zde jsou popsány hlavní třídy herního jádra. Herní jádro implementuje většinu funkcí nutných pro realizaci hry (tj. provádění jednotlivých kol a fází). Herní jádro potom nezprostředkovává vlastní interakci s hráči ani s počítačovými hráči. Interakce s hráči poskytuje uživatelské rozhraní obsažené v RoboRally GUI (s počítačovými hráči i RoboRally_Console). Herní jádro je využíváno aplikacemi RoboRally GUI a RoboRally_Console.

B.1.1 Datové struktury herní logiky

Hlavní třídou, která reprezentuje celý stav hry a umožňuje provádět programy robotů je třída `GameCore`. Tato třída obsahuje reprezentaci herního plánu v instanci třídy `Map`, jednotlivé roboty ve hře v instancích třídy `Robot` a balíček karet v instanci třídy `CardPackage<MotionCard>`. Kromě těchto datových položek musí `GameCore` obsahovat ještě kód provádějící jednotlivé registry robotů. K tomu obsahuje rozhraní `IPhase`:

```
private interface IPhase {
     void PrepareExecution(int register);
     bool ExecutePhase(int register);
     string GetName();
     string GetIdentifier();
     Position<> GetAffectedPositions();
```

```
    Robot GetAffectedRobot();
}
```

Pro každý krok provádění registrů je v programu využita jedna implementace rozhraní `IPhase`. Třída `GameCore` obsahuje pevný seznam instancí tříd implementujících toto rozhraní, které dohromady tvoří velkou část pravidel hry. Díky tomuto rozhraní je ale možné tento seznam snadno upravit a například do hry přidat další krok, který potom bude v každém kole prováděn. Tak je umožněna snadná změna pravidel hry, kterou někteří pokročilí hráči využívají (u stolních verzí hry). Implementace rozhraní vytvořené v aktuální verzi jsou následující:

1) `MovePhase` – implementuje vykonávání karet v registrech robotů.
2) `ConveyorBeltPhase` – implementuje pohyby dopravníků.
3) `PusherPhase` – implementuje pohyb strkačů.
4) `GearPhase` – implementuje otáčení převodovek.
5) `LaserPhase` – implementuje střelbu laserů.
6) `TouchPhase` – implementuje dotyk vlajek a opraven.

Před každým kolem je nutné naprogramovat všechny roboty (definovat obsah registrů), to třída `GameCore` nezajišťuje, tuto informaci obdrží jako součást jednotlivých robotů. Následně je pro vykonání jednoho kola hry nutné provést inicializaci nového kola (metoda `InitRound`) a potom postupně volat `ExecuteNextPhase`, čímž se budou provádět jednotlivé fáze hry. Mezi fázemi se lze různými metodami třídy `GameCore` dotazovat, co bude následující fáze měnit a upozorňovat na to hráče.

Jednotlivé roboty ve hře reprezentuje třída `Robot`. Ta uchovává všechna data specifická pro roboty, umožňuje jejich ukládání a načítání ze souboru, pohyb robota po hracím plánu, jeho otáčení a podobně. Balíček karet ve hře je reprezentován třídou `CardPackage`, její hlavní metody umožňují odebrání karty z balíčku a její navrácení a zamíchání celého balíčku. Jednotlivé karty reprezentuje abstraktní třída `MotionCard` a její podtřídy `MoveCard` a `RotationCard`. Balíček karet je vždy načítán ze souboru, to provádí statická metoda `LoadPackageFromFile` třídy `MotionCard`.

Herní plán reprezentuje třída `Map`. Instance třídy `Map` je při započetí hry vytvářena z instance třídy `BoardField` (ta umožňuje sestavení herního plánu). Herní plán se skládá z jednotlivých políček, která jsou ve třídě `Map` uložena ve

dvourozměrném poli, které má pro jednoduchost vždy stejnou velikost (ta je dána maximální velikostí herního plánu, který lze vytvořit v `BoardField`). Třída `Map` umožňuje vytvoření obrázku herního plánu a provádění různých dotazů nad ním.

Jednotlivá políčka jsou reprezentována třídou `BoardItem`. Tato třída sama v sobě obsahuje datové položky společné pro všechny druhy políček, tj. obrázky, seznam prvků na políčku, stěna, a také metody, které využívají všechna políčka, tj. vykreslení, klonování, otočení políčka o 90°(využívá se při sestavování herního plánu). Specifická políčka pak potřebují vlastní datové položky, případně metody, (nebo jen odlišení pomocí typu) proto vlastní políčka reprezentují až její podtřídy:

1) `ConveyorBelt` – podtřída pro dopravníky a expresní dopravníky.
2) `BottomlessPit` – podtřída pro bezednou jámu.
3) `Ordinary` – podtřída pro obyčejné políčko.
4) `Gear` – podtřída pro převodovku.
5) `RepairSite` – podtřída pro opravnu a vylepšující opravnu.

Stěnu, která může být na každém políčku, reprezentuje třída `Wall`. Každé políčko má nejvýše jednu instanci třídy `Wall`, ta představuje všechny stěny kolem celého políčka (může představovat jejich libovolnou kombinaci). Ostatní prvky, které mohou být přítomny na políčku, reprezentuje třída `BoardElement`. Ta má nasledující podtřídy, reprezentující jednotlivé prvky:

1) `Laser` – podtřída pro laser.
2) `Pusher` – podtřída pro strkač.
3) `Flag` – podtřída pro vlajku.
4) `StartPos` – podtřída pro startovní pozici.

B.1.2 Datové struktury umělé inteligence

Každý robot ve hře může být ovládán počítačem. (Část programu ovládající robota pak dále nazývám engine.) K tomu má vlastnost `Engine`, do které lze přiřadit instanci třídy implementující rozhraní `IRobotEngine`. Toto rozhraní specifikuje různé metody, kterými je engine vyzýván k rozhodování o chování svého robota. Více tuto problematiku rozvádí kapitola *4 Vytvoření počítačového hráče*.

B.1.3 Datové struktury určené k sestavení herního plánu

Sestavování herního plánu umožňují třídy `BoardField` a `Board`. Třída `Board` reprezentuje jednotlivé desky, ze kterých lze hrací plán sestavit, a poskytuje k nim operace otočení, načtení ze souboru a vykreslení desky. Desky ve hře jsou pro jednoduchost vždy čtvercové velikosti o hraně dvanáct políček. Jednotlivá políčka pak jsou reprezentována třídou `BoardField`, stejně jako ve třídě `Map`.

Třída `BoardField` umožňuje sestavování herních plánů. Herní plán lze buď načíst ze souboru, nebo jej vytvořit v grafickém rozhraní, které `BoardField` poskytuje. Vytvořené herní plány, je pak možné ukládat do souborů.

B.2 RoboRally GUI

Program RoboRally GUI k hernímu jádru přidává grafické rozhraní interagující s hráčem. Interakce s hráči obsahuje:

1) Sestavení herního plánu.
2) Rozdání karet každému hráči a umožnění hráčům naprogramovat roboty.
3) Umožnění hráčům navrátit robota do hry v případě, že robot zemře.
4) Umožnit hráči použít štíty robota, pokud je to možné.
5) Omezit čas na programování robota. Tento čas není omezen pro počítačové hráče, ti jsou dostatečně rychlí.

Grafické rozhraní programu RoboRally GUI poskytuje všechny potřebné ovládací prvky k vizualizaci hry RoboRally, která je implementována herním jádrem. K vytvoření uživatelského rozhraní byla použita knihovna Windows Forms. K sestavení uživatelského rozhraní bylo použito již existujících komponent z Windows Forms. Bylo však nutné i vytvoření nových komponent specielně pro hru. Tyto komponenty byly většinou vytvořeny jako podtřídy třídy `Control`, díky čemuž dobře zapadají do již existujícího systému komponent. Vytvořeno bylo i několik formulářů, jako podtřídy třídy `Form`.

B.2.1 Drag and Drop

„Drag and drop“, česky „táhni a pusť“, znamená chycení nějakého prvku, který umožňuje „uchopení“, tím že nad ním stiskneme tlačítko myši a držíme. Pak přesuneme kurzor myši nad jiný prvek uživatelského rozhraní, který umožňuje operaci „upustit“, nad ním tlačítko myši uvolníme. Tím se mezi danými prvky (může to být i jeden a ten samý prvek) přenesou nějaká data. Tento postup umožňuje

uživatelsky přívětivý způsob nastavování informací, přesouvání dat, nebo posouvání obrázků nebo grafický prvků. Řada komponent uživatelského rozhraní RoboRally GUI umožňuje realizaci příslušné akce nebo nastavení právě metodou „drag and drop“.

B.2.2 GUI komponenty

Zde jsou popsány nejdůležitější vytvořené komponenty uživatelského rozhraní. Základními komponentami vytvořenými pro realizaci uživatelského rozhraní programu jsou:

1) `PictureViewer` – komponenta umožňující zobrazení obrázku, který má větší rozměry než komponenta samotná. Zobrazen je vždy jen výřez z obrázku, který se do komponenty aktuálně vejde. Na okrajích komponenty jsou posuvníky, ukazující, která část obrázku je právě viditelná. Je možné měnit, která část obrázku je právě zobrazena, a to buď metodou „drag and drop“ přímo nad obrázkem, čímž myší posouváme obrázek, nebo nad posuvníky, ty se chovají stejně jako standardní posuvníky ve Windows.
2) `MapViewer` – komponenta, která zobrazuje herní plán. K tomuto účelu využívá instanci PictureViewer zobrazující obrázek herní mapy daný instancí třídy Map. MapViewer umožňuje volbu zvětšení mapy (úrovně velikosti mapy).
3) `MiniMap` – komponenta zobrazující hrubou zmenšeninu herního plánu. Slouží k lepší orientaci na herním plánu. Na ní nejsou vidět jednotlivá políčka, ale jen jednotlivé desky hracího plánu, zvýrazněna jsou pouze místa, na kterých se vyskytují roboti. Pro správnou funkci potřebuje mít instance `MiniMap` k dispozici instanci třídy `MapViewer`, potom zvýrazňuje, která část mapy je právě v dané instanci `MapViewer` zobrazena. Naopak metodou „drag and drop“ lze určit, která část mapy má být v `MapVieweru` zobrazena.
4) `RobotList` – komponenta podobná třídě `ListBox` z Windows Forms. Zobrazuje seznam robotů a umožňuje některého z nich vybrat. Pro každého robota zobrazuje jeho obrázek, jméno robota i jeho hráče a základní informace: následující vlajku, počet životů, počet poškození a počet štítů.
5) `RobotPanel` – komponenta zobrazující informace aktuálně přiřazeného robota.
6) `RegisterPanel` – komponenta zobrazující určitý registr seznamu robotů.

7) `BoardField` – komponenta umožňující sestavení herního plánu, nebo jeho načtení ze souboru.
8) `CardHolder` – komponenta zobrazující kartu pohybu. Může mít přiřazenou buď pouze kartu pohybu, nebo registr nějakého robota, potom se zobrazovaná karta převezme z daného registru. `CardHolder` se chová různými způsoby, v závislosti na nastavení (odemknutý/zamknutý, schovaný/neschovaný). Mezi odemčenými, neschovanými `CardHoldery` lze jejich karty přetahovat metodou „drag and drop“. Základní modifikace nastavení jsou:
   a) Zamknutý – pak s kartou v `CardHolderu` nelze hýbat, ale je zobrazena. Pokud má `CardHolder` přiřazen registr robota, pak to, jestli bude zamčený, se určí z nastavení daného registru.
   b) Schovaný – pak není vidět jestli, nebo jakou kartu má `CardHolder` přiřazenu a s případnou kartou nelze hýbat.

B.3 RoboRally_Console

Program RoboRally_Console je konzolová aplikace umožňující hraní her mezi různě nastavenými enginy. K vytvoření hry program využívá třídy herního jádra. Bližší popis funkcí tohoto programu je v kapitole *5 Nastavení parametrů pomocí genetických algoritmů*.

B.4 Formáty souborů RoboRally

V této kapitole jsou popsány adresářová struktura a soubory nutné pro běh aplikací RoboRally a RoboRally_Console. Formáty souborů a vyžadovaná adresářová struktura jsou společné pro RoboRally a RoboRally_Console. RoboRally_Console se liší v tom, že neumožňuje uložení a načtení uložené hry a nemá grafické uživatelské rozhraní, tedy nepotřebuje některé obrázky a adresář pro uložené hry. Přestože se ale mapa ani karty nikde nevykreslují, jsou jejich obrázky nutné již pro jejich vytvoření (toto chování by asi bylo dobré změnit), proto jsou tyto obrázky potřeba.

Tři typy souborů jsou v textovém formátu, díky tomu není nutné mít pro soubory s kartami a soubory s deskami zvláštní programy pro jejich vytváření a editaci (soubory s hracími plány jsou sice textové, aby šlo i herní plány vytvářet bez specielního programu, ale ty lze vytvářet v aplikaci RoboRally GUI). Soubory

s uloženými hrami jsou binární, takto je snazší tyto soubory ukládat a načítat (nemusejí se parsovat, formát je přesně daný).

Textové soubory jsou ukládány v kódování UTF-8. V těchto souborech obsahují data jen řádky začínající „*#*“, ostatní řádky jsou ignorovány (brány jako komentář). V následujícím popisu formátu textových souborů „*\t*“ znamená tabelátor. Písmena *u*, *l*, *d*, *r* v následujícím určují směr, nebo umístění některých akcí/prvků, jejich význam je: *u* – nahoru/nahoře, *l* – doleva/vlevo, *d* – dolů/dole, *r* – doprava/vpravo.

B.4.1 Adresářová struktura

Zde jsou popsány jednotlivé adresáře a jejich obsah, které programy RoboRally GUI a RoboRally_Console potřebují. Všechny uvedené adresáře musejí být v adresáři, kde je daný program (v aktuálním adresáři tohoto programu po jeho spuštění). (Každý obrázek, který je používán při vykreslování mapy, tedy obrázky políček, robotů, laserů, strkačů a zdí je obsažen v šesti velikostech, používaných pro různá zvětšení mapy.)

1) boards – Adresář obsahující soubory s definicemi desek, ze kterých se skládají herní plány.
2) cards – Adresář s obrázky všech karet pohybu, dalšími obrázky používanými třídou CardHolder a souborem s definicemi karet pohybu.
3) img – Adresář obsahující další adresáře, které odpovídají různým políčkům na deskách. Tyto adresáře obsahují soubory „0.png“ až „5.png“ s různými velikostmi obrázků políček.
4) imgs – Adresář obsahující obrázky používané třídou RobotPanel. (Tento adresář nepoužívá RoboRally_Console.)
5) laser – Adresář obsahující obrázky laserů.
6) maps – Adresář obsahující soubory s uloženými hracími plány.
7) pusher – Adresář obsahující obrázky strkačů.
8) robots – Adresář obsahující pro každého robota adresář s jeho obrázky.
9) savedgames – Adresář s uloženými hrami. (Tento adresář nepoužívá RoboRally_Console.)
10) wall – adresář s obrázky stěn.

B.4.2 Soubor s deskou

Soubor s deskou je textový soubor. Políčka desky, která nejsou správně definována, nebo nejsou vůbec v souboru zadána, jsou vykreslena černě a chovají se jako bezedné jámy. Soubor s deskou obsahuje nejprve definice políček v ní použitých:

#D\t<jméno>\t<typ políčka> <parametry políčka>\t<cesta>

Jednotlivé části záznamu:

1) *<jméno>* – jedno slovo bez bílých znaků, které se používá ke vkládání tohoto políčka do mapy.
2) *<typ políčka> <parametry políčka>* – určuje jakého typu políčko je. Daný typ může a nemusí mít parametry, pokud nějaké parametry má, pokud má políčko více parametrů, jsou od sebe odděleny mezerou. Jednotlivé typy jsou:
   a) *1* – obyčejné políčko
   b) *2* – obyčejný dopravník. Má dva parametry, které se od sebe oddělují mezerou:
      i) *u|l|d|r* – jednopísmenný parametr, určující směr, kterým je robot z tohoto políčka dopravníkem posunut (výstupní směr políčka).
      ii) *uldr* – parametr, obsahující alespoň jedno ze zadaných písmen (ale žádné dvakrát), určující směr nebo směry, ze kterých pokud je robot na dané políčko posunut dopravníkem, je otočen do směru určeného prvním parametrem (vstupní směry políčka).
   c) *3* – expresní dopravník. Má stejné parametry jako normální dopravník.
   d) *4* – převodovka. Má jeden jednopísmenný parametr, určující jak bude robot převodovkou otočen:
      i) *r* – znamená, že robot bude otočen o 90° ve směru hodinových ručiček.
      ii) *l* - znamená, že robot bude otočen o 90° proti směru hodinových ručiček.
   e) *5* – opravna. Má jeden parametr určující, jestli robot, který na opravně stojí na konci kola, dostane kartu s vylepšením. Tento parametr může nabývat hodnot:
      i) *t* – robot, který na opravně stojí na konci kola, dostane kartu s vylepšením.
      ii) *f* – robot nedostane kartu s vylepšením.
   f) *6* – bezedná jáma, nemá žádné parametry.

3) <*cesta*> – název složky v adresáři „./imgs“, která obsahuje obrázky pro tento druh políčka.

Dále soubor obsahuje záznamy řádků desky, záznam každého řádku začíná *„#R\t“* a následují záznamy jednotlivých políček pro daný řádek oddělené od sebe *„\t“*. Formát záznamů jednotlivých políček je následující:

X <jméno> <zdi> <strkač><fáze> <lasery>

Kde jednotlivé části záznamu jsou:

1) <*jmeno*> – jméno nějakého dříve definovaného políčka. Určuje, jak se bude políčko chovat, a jak bude vypadat.
2) <*zdi*> – je nepovinná část. Skládá se z písmene *W*, následovaným alespoň jedním písmenem z: *uldr*. Určuje, na kterých stranách políčka jsou zdi.
3) <*strkac*><*faze*> – je nepovinná část. <*strkac*> se skládá z písmene P, následovaným právě jedním písmenem z: uldr. Určuje, na které straně políčka je strkač, na dané straně musí být také nastavena zeď. <*faze*> je seznam čísel z rozmezí jedna až pět (každé číslo nejvýše jednou), určující, ve kterých fázích vykonávání registrů robotů je strkač aktivní.
4) <*lasery*> – je nepovinná část. Skládá se z písmene *L*, následovaným alespoň jedním písmenem z: *uldr*, přičemž každé písmeno může být zadáno až třikrát. Určuje, na kterých stranách políčka je kolik laserů. Na každé straně políčka, kde je alespoň jeden laser, musí být také nastavena zeď.

Nakonec soubor obsahuje záznam se jménem a může obsahovat i záznamy určující rozmístění vlajek na desce, nebo rozmístění startovních pozic na desce:

#N “<jméno>“ - znamená, že název desky je <*jméno*>.

#F <číslo vlajky> <x>,<y> – znamená, že vlajka <*číslo vlajky*> je na desce umístěna na políčku se souřadnicemi <*x*>,<*y*>. <*cislo vlajky*> může nabývat hodnot jedna až osm. Obě souřadnice jsou v rozmezí jedna až dvanáct a určují v kolikátém sloupci (<*x*>) zleva, a v kolikáté řádce (<*y*>) odshora se vlajka nachází.

#S <číslo startovní pozice> <x>,<y> - znamená, že startovní pozice <*číslo startovní pozice*> je na desce umístěna na políčku se souřadnicemi <*x*>,<*y*>. Rozmezí pro daná čísla jsou stejná jako pro záznam s vlajkou.

B.4.3 Soubor s hracím polem

Soubor obsahuje záznam se jménem a případně záznamy vlajek a startovních pozic stejně jako soubor s deskou. Dále obsahuje tři záznamy začínající *„#R\t“*,

každý z nich pokračuje třemi záznamy desek hracího pole oddělenými *„\t“*, a určuje tak jeden řádek matice hracího pole. Záznamy jednotlivých desek mají následující formát:

B “<jméno>“ <číslo>

Parametry *<jméno>* a *<číslo>* jsou nepovinné, pokud nejsou obsaženy, dané místo herního pole neobsahuje desku. Pokud jsou, pak *<jméno>* určuje název hrací desky, která je na daném místě hracího pole a *<číslo>* je číslo určující kolikrát je deska otočena o 90° ve směru hodinových ručiček.

B.4.4 Soubor s kartami pohybu

Pro jednotlivé druhy instrukcí (typy karet) obsahuje soubor následující záznamy:

#<typ karty>\t“<soubor>“\t<priorita1>,<priorita2>,...

Význam jednotlivých částí záznamu:

1) *<typ karty>* – číslo určující instrukci karty. Instrukce mohou být následující:
   a) *1* – otočení vzad.
   b) *2* – otočení doleva.
   c) *3* – otočení doprava.
   d) *4* – krok vzad.
   e) *5* – krok vpřed.
   f) *6* – dva kroky vpřed.
   g) *7* – tři kroky vpřed.
2) *“<soubor>“* - cesta k obrázku daného typu karty v uvozovkách, pokud by soubor obsahoval více záznamů se stejným typem karty, použije pro daný typ karty obrázek, který určí poslední takový záznam v souboru.
3) *<priorita1>,<priorita2>* – seznam záznamů priorit, záznam může být:
   a) *číslo* – pak se do balíčku karet hry přidá karta typu *<typ karty>* s prioritou danou tímto číslem.
   b) *priorita1..priorita2-krok* – pak se do balíčku přidají karty typu *<typ karty>* s prioritami danými vzorcem: priorita1 + i * krok, pro i >= 0 a priorita1 + i * krok <= priorita2.

B.4.5 Soubor s uloženou hrou

Soubory s uloženými hrami jsou binární, aby bylo snazší je ukládat a číst, jejich čtení nebo úprava uživateli nejsou potřeba. Soubory jsou rozděleny do bloků (ke

čtení, ukládání a dalším operacím s těmito bloky v programu slouží třída FileRecord, která do bloku umí vkládat všechny základní typy). Každý blok obsahuje:

1) Typ – 4B integer, udávající typ bloku.
2) Velikost – 4B integer, udávající počet datových bajtů bloku.
3) Data
4) Kontrolní součet – 4B integer, součet všech datových bajtů.

Všechny ukládané bloky mají jako typ nulu, jakého je blok typu se rozhoduje podle pořadí v souboru a informací z předchozích bloků. Soubor s uloženou hrou se skládá z bloků s následujícími daty:

1) Název hry (string).
2) Hrací plán, ve formátu jako je v souboru s uloženým hracím plánem (string).
3) Čas na programování robota (4B string), počet robotů ve hře (4B string), počet robotů, kteří úspěšně dokončili hru (4B string), počet robotů, kteří neúspěšně skončili hru (4B string).
4) Bloky s postupně uloženými hrajícími roboty, roboty, kteří úspěšně dokončili hru a roboty, kteří neúspěšně skončili hru.

B.5 Evolution2

Program Evolution2 konzolová aplikace umožňující hledání vhodného nastavení zvoleného enginu pomocí techniky genetických algoritmů. Popis této aplikace a jejích funkcí je v kapitole *5 Nastavení parametrů pomocí genetických algoritmů*.

Jedinci populace (natavení enginu) jsou reprezentování třídou `EngineSettings`. Tato třída umožňuje generovat náhodné jedince (`GenerateRandomParams`), uložit jedince do souboru (`SaveToFile`) nebo jej z něj načíst (`LoadFromFile`), vytvořit nového jedince zkřížením dvou jedinců (`Cross`) a mutaci jedince (`Mutate`). Instance třídy si pamatuje všechny parametry (geny) jedince, jeho získané ohodnocení, soubor v jakém je uložen (ten je nutný pro použití jedince ve hře pomocí RoboRally_Console), číslo jedince v populaci, jméno jedince (tím je jednoznačně identifikován v záznamu) a počet her, které sehrál (používá se při generování her). Populace je v programu reprezentována jednoduše seznamem jedinců, žádná speciální datová struktura pro ni není.

Pro ohodnocování jedinců jsou mezi nimi sehrávány hry pomocí programu RoboRally_Console. Kvůli umožnění využití více procesorů jsou hrané hry nejprve vygenerovány do seznamu, ze kterého jsou následně vybírány volnými vlákny

a prováděny. Díky předgenerování seznamu s hrami stačí, aby byl „thread safe“, pouze seznam s těmito hrami, který je používán více vlákny. Generování hraných her „thread safe“ být nemusí (a ani nic jiného), to a ani nic jiného ani není potřeba provádět na více procesorech, jelikož oproti hraní her jsou to časově zanedbatelné operace.

Hra, která se má odehrát, i její výsledek jsou reprezentovány třídou `Game`. Tato třída uchovává jednotlivé jedince účastnící se hry, body za získané pozice ve hře, výsledky jedinců v odehrané hře, název mapy, na které se hra odehrává a jako dlouho hra trvala, jedinou její důležitou metodou je vytvoření řetězce s informacemi o hře (`ToString`). Seznam her reprezentuje třída `Games`, jejíž všechny metody jsou „thread safe“, její hlavní metody umožňují přidání nové hry do seznamu (`AddGame`) a získání další ještě neodehrané hry (`GetNextUnplayedGame`). K odehrání seznamu her reprezentovaného v `Games` slouží třída `GamePlayer`. Tato třída umí seznam her odehrát s využitím více vláken pomocí metody `PlayGames`. Jednotlivá vlákna jsou reprezentována její vnitřní třídou `GameThread`.

Pro zaznamenávání činnosti programu byla vytvořena třída `Log`. Ta umožňuje vypisovat text na konzoli a zároveň do souboru.

Kód řídící běh evoluce je obsažen především ve statických metodách třídy Program, nastavení běhu pak především v jejích statických datových položkách. Hlavní cyklus evoluce (obsahující ohodnocování jedinců populace a generování nové populace) je obsažen v metodě `Main`. K uložení a načtení populace slouží metody `SavePopulation` a `LoadPopulation`. K vygenerování her v rámci populace za účelem ohodnocení jedinců slouží metoda `PlayGames` a k vygenerování her porovnávajících populace metody `MatchPopulations` a `MatchPopulationWith`.

Příloha C: Obsah přiloženého CD

Níže je přibližně popsána adresářová struktura přiloženého CD včetně popisu důležitých souborů a složek:

1) Složka „experimenty“ – obsahuje složky provedených běhů a pro každý běh skript (skript pro Windows Powershell 2.0), který stejný běh vytváří (tyto skripty musejí být spouštěny ve složce, kde je nainstalováno RoboRally).
2) Složka „instalace“ – Obsahuje instalaci programu RoboRally. (S ním jsou instalovány i RoboRally_Console a Evolution2.)
3) Složka „projekt“ – Obsahuje projekt pro Visual Studio 2010, obsahující všechny tři vytvořené aplikace. Pro každou z aplikací je zde obsažena další složka, v níž jsou její zdrojové kódy.
4) Složka „text prace“ – Obsahuje elektronickou verzi této práce ve formátu PDF.